US 20190147981A1

(19) **United States**

(12) **Patent Application Publication**

van Rooyen et al.

(10) **Pub. No.: US 2019/0147981 A1**

(43) **Pub. Date: May 16, 2019**

(54) **BIOINFORMATICS SYSTEMS, APPARATUSES, AND METHODS EXECUTED ON AN INTEGRATED CIRCUIT PROCESSING PLATFORM**

(71) Applicant: **Edico Genome, Corp.**, La Jolla, CA (US)

(72) Inventors: **Pieter van Rooyen**, La Jolla, CA (US); **Michael Ruehle**, La Jolla, CA (US); **Robert McMillen**, La Jolla, CA (US); **Mark Hahm**, San Diego, CA (US)

(21) Appl. No.: **16/119,976**

(22) Filed: **Aug. 31, 2018**

Related U.S. Application Data

(63) Continuation of application No. 15/059,221, filed on Mar. 2, 2016, now Pat. No. 10,068,054, which is a continuation-in-part of application No. 15/048,935, filed on Feb. 19, 2016, which is a continuation-in-part of application No. 14/284,307, filed on May 21, 2014, now Pat. No. 9,235,680, which is a continuation of application No. 14/279,063, filed on May 15, 2014, now Pat. No. 9,679,104, which is a continuation-in-part of application No. 14/158,758, filed on Jan. 17, 2014, now Pat. No. 9,483,610.

(60) Provisional application No. 62/127,232, filed on Mar. 2, 2015, provisional application No. 62/119,059, filed on Feb. 20, 2015, provisional application No. 61/988,128, filed on May 2, 2014, provisional application No. 61/984,663, filed on Apr. 25, 2014, provisional application No. 61/943,870, filed on Feb. 24, 2014, provisional application No. 61/910,868, filed on Dec. 2, 2013, provisional application No. 61/823,824, filed on May 15, 2013, provisional application No. 61/822,101, filed on May 10, 2013, provisional application No. 61/826,381, filed on May 22, 2013, provisional application No. 61/753,775, filed on Jan. 17, 2013, provisional application No. 62/127,232, filed on Mar. 2, 2015.

Publication Classification

(51) **Int. Cl.**

G16B 40/00	(2006.01)
G16B 50/00	(2006.01)
G06N 3/00	(2006.01)
H03K 19/177	(2006.01)
G16B 30/00	(2006.01)

(52) **U.S. Cl.**

CPC ***G16B 40/00*** (2019.02); ***G16B 50/00*** (2019.02); ***G16B 30/00*** (2019.02); ***H03K 19/17736*** (2013.01); ***G06N 3/002*** (2013.01)

(57) **ABSTRACT**

A system, method and apparatus for executing a sequence analysis pipeline on genetic sequence data includes an integrated circuit formed of a set of hardwired digital logic circuits that are interconnected by physical electrical interconnects. The hardwired digital logic circuits are arranged as a set of processing engines, each processing engine being formed of a subset of the hardwired digital logic circuits to perform one or more steps in the sequence analysis pipeline on the reads of genomic data. In various instances, each subset of the hardwired digital logic circuits may be formed in a wired configuration to perform one or more steps a variant call operation.

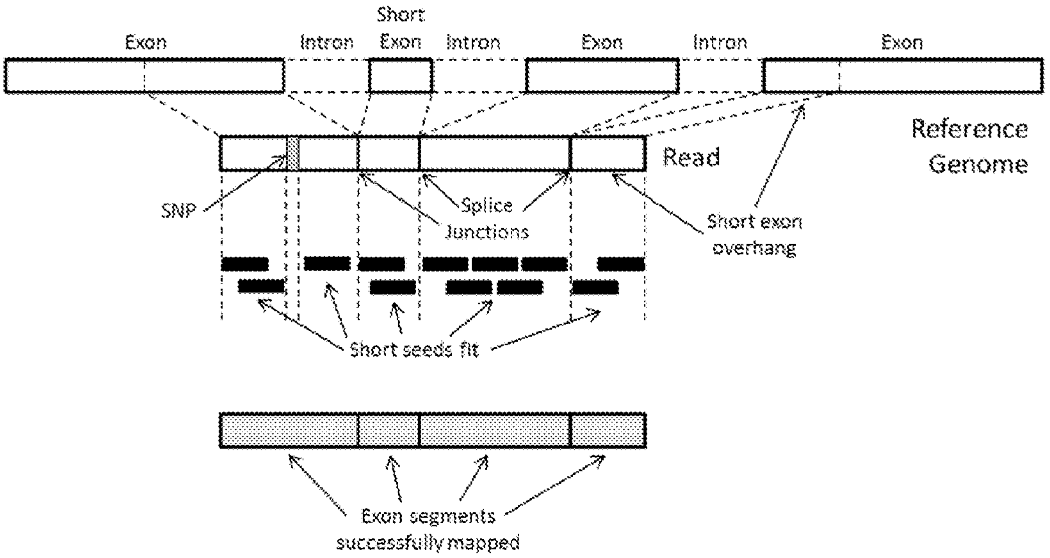


FIG. 1

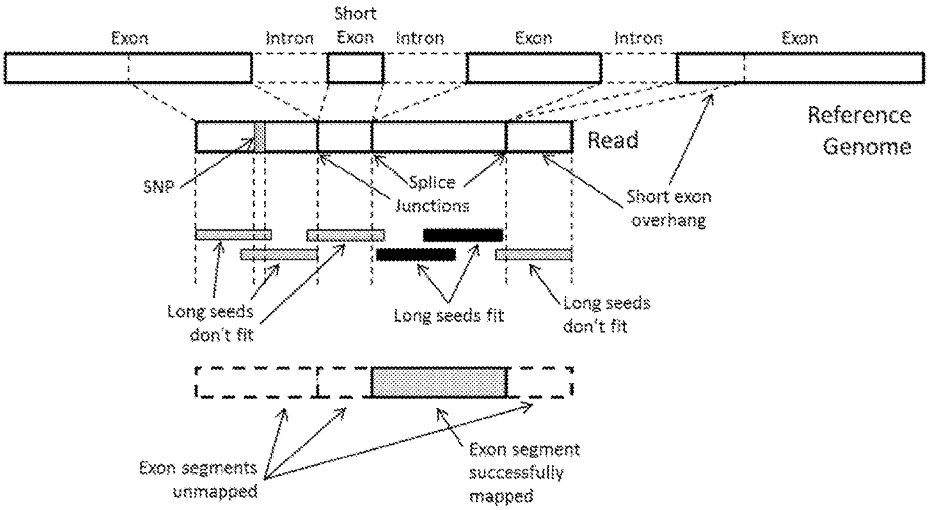

Exon
Intron
Short Exon
Intron
Exon
Intron
Exon
Reference Genome
Read
SNP
Splice Junctions
Short exon overhang
Long seeds don't fit
Long seeds fit
Long seeds don't fit
Exon segments unmapped
Exon segment successfully mapped

FIG. 2

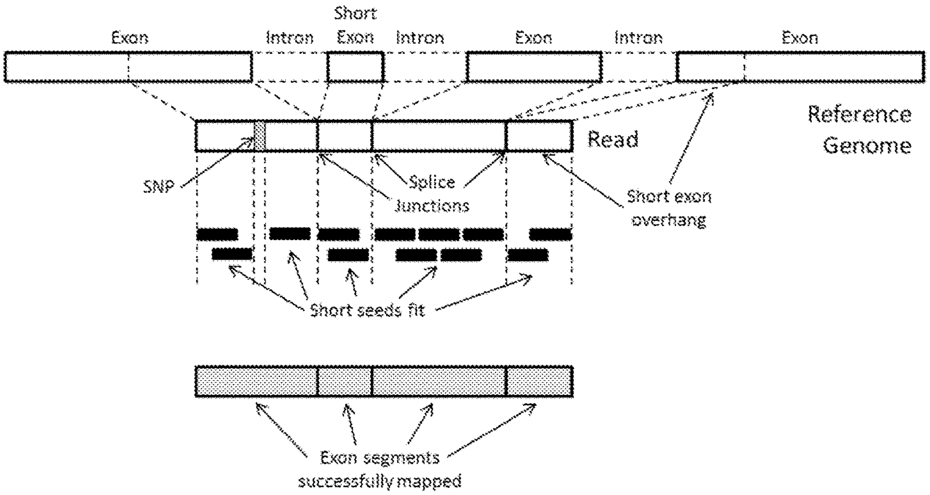

Exon
Intron
Short Exon
Intron
Exon
Intron
Exon
Reference Genome
Read
SNP
Splice Junctions
Short exon overhang
Short seeds fit
Exon segments successfully mapped

FIG. 3

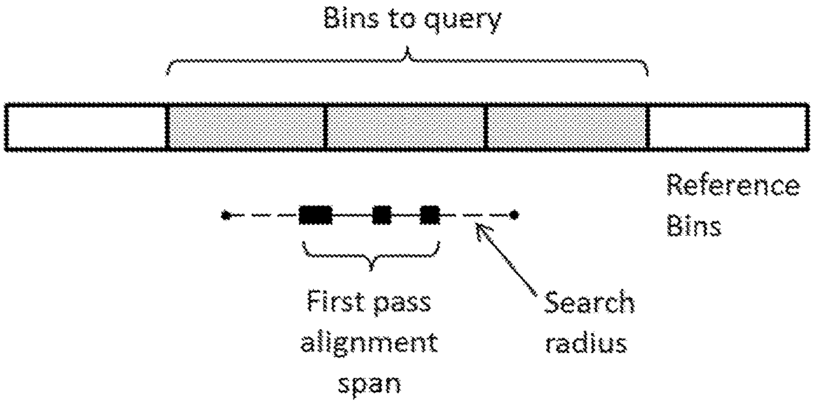

Bins to query
Reference Bins
First pass alignment span
Search radius

FIG. 4

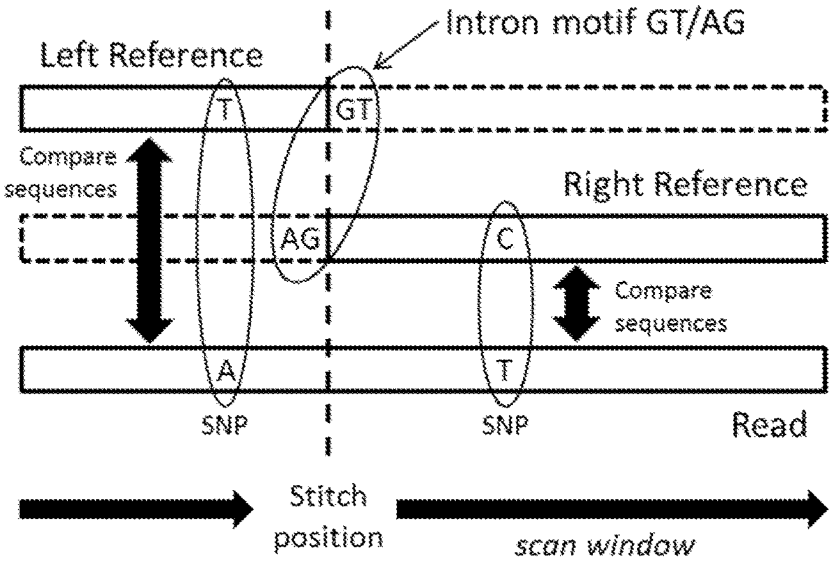

Intron motif GT/AG
Left Reference
T
GT
Compare sequences
Right Reference
AG
C
Compare sequences
A
T
SNP
SNP
Read
Stitch position
scan window

FIG. 5

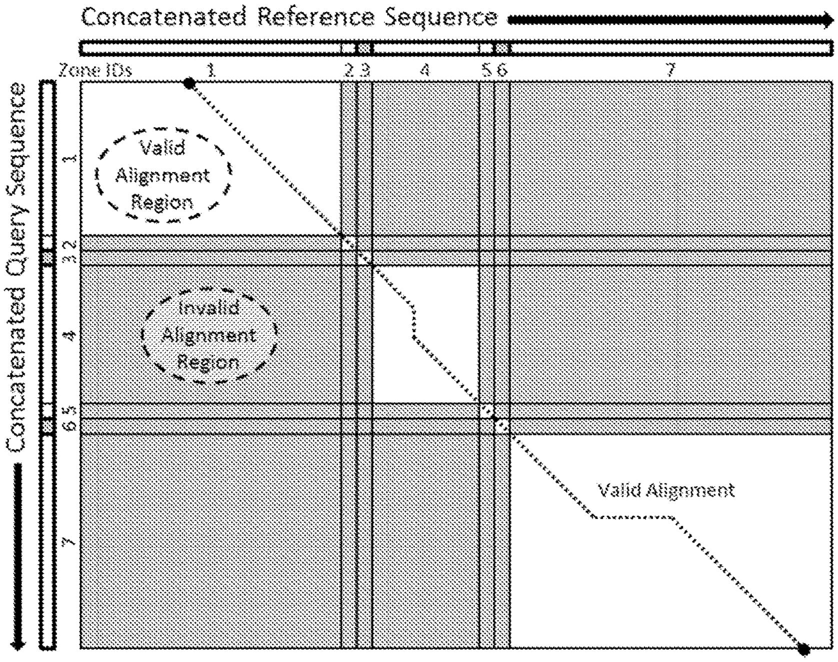

Concatenated Reference Sequence
Zone IDs
1
2 3
4
5 6
7
Concatenated Query Sequence
1
2
3
4
5
6
7
Valid Alignment Region
Invalid Alignment Region
Valid Alignment

FIG. 6

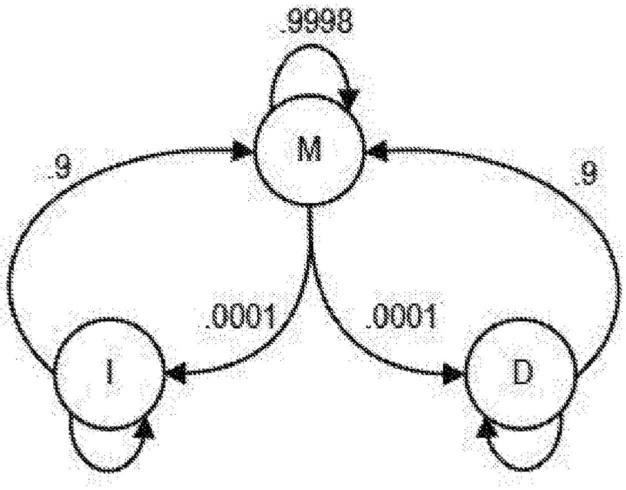

.9998
M
.9
.9
.0001
.0001
I
D

FIG. 7

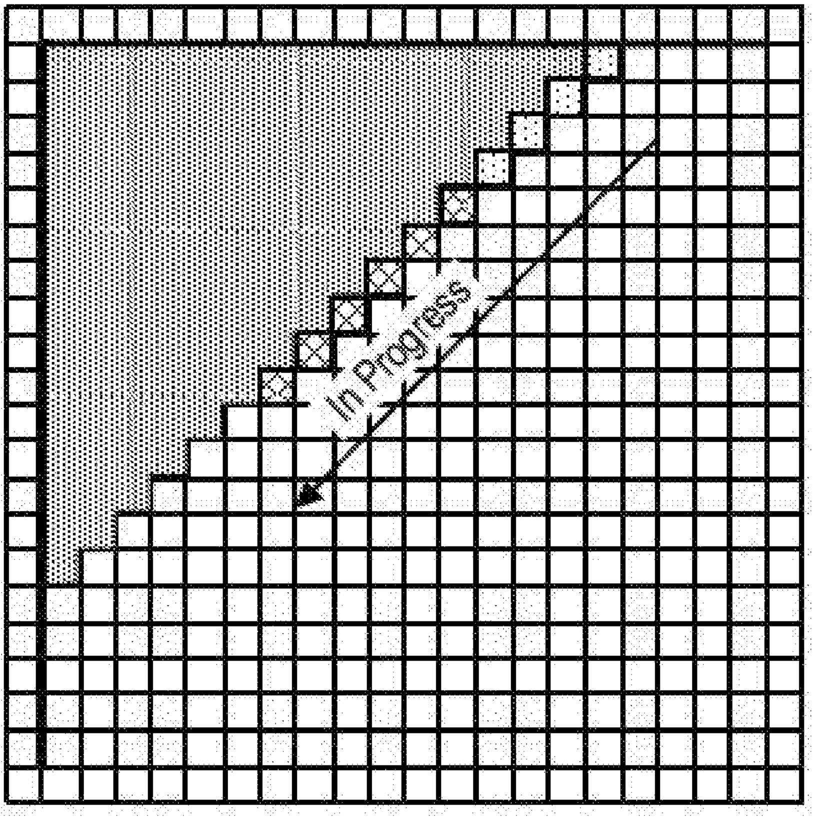


FIG. 8

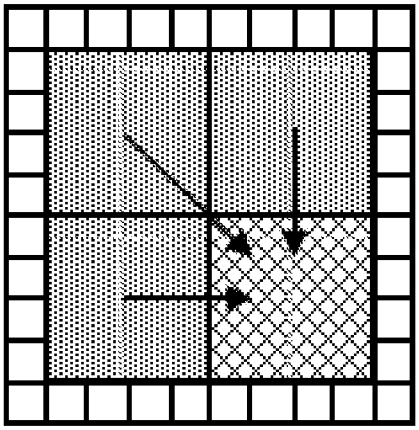


FIG. 9

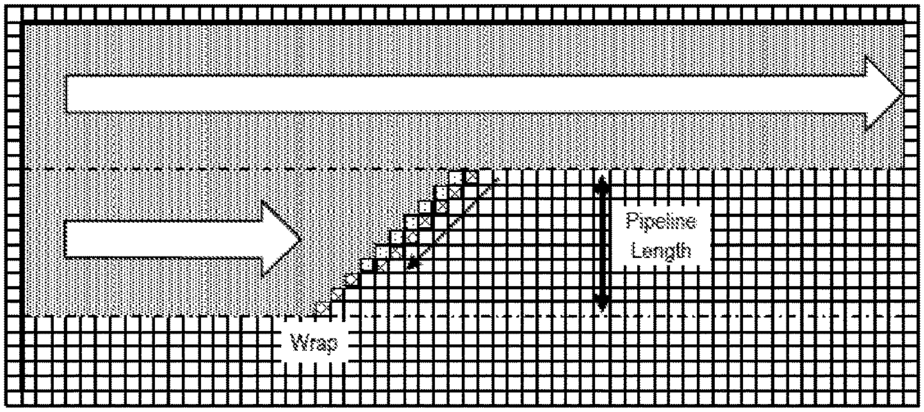

Pipeline Length
Wrap

FIG. 10

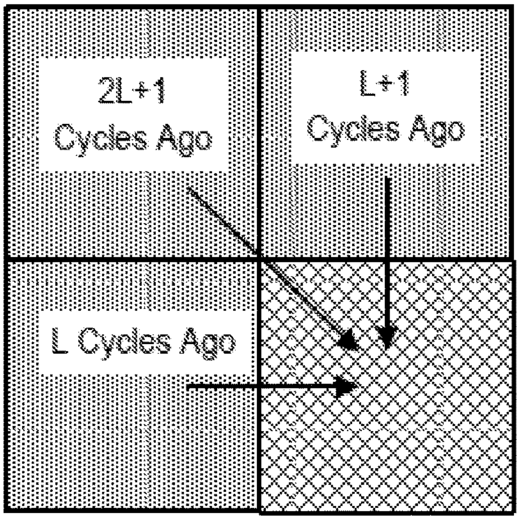

2L+1 Cycles Ago
L+1 Cycles Ago
L Cycles Ago

FIG. 11

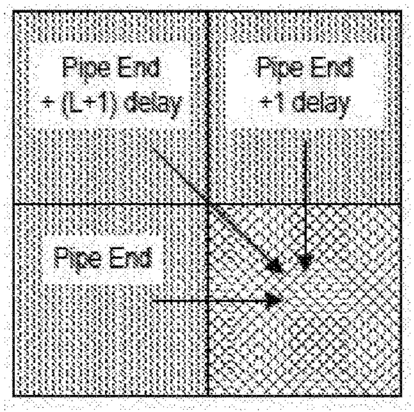

Pipe End
+ (L+1) delay
Pipe End
+1 delay
Pipe End

FIG. 12

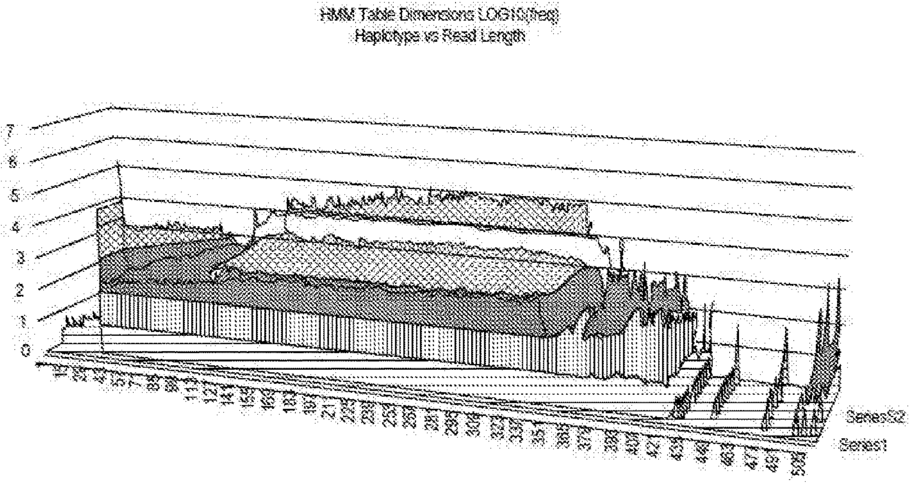
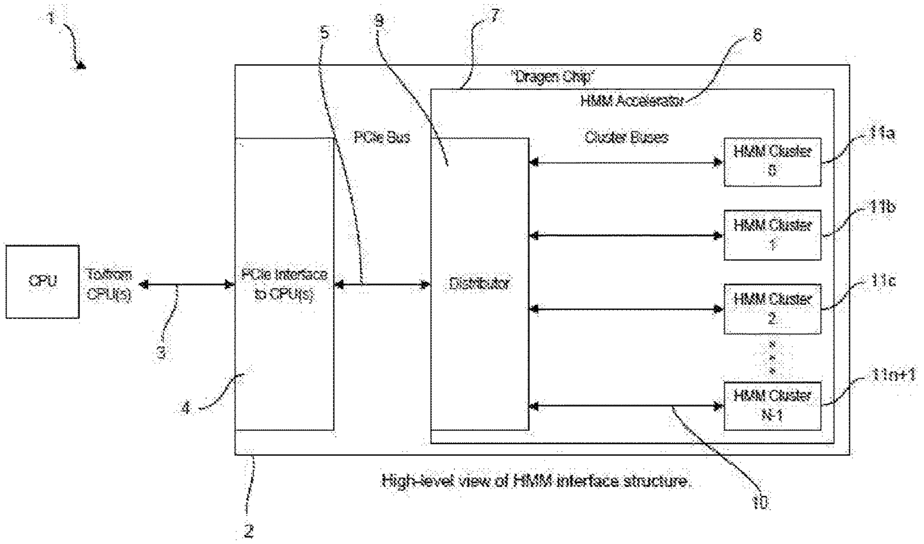
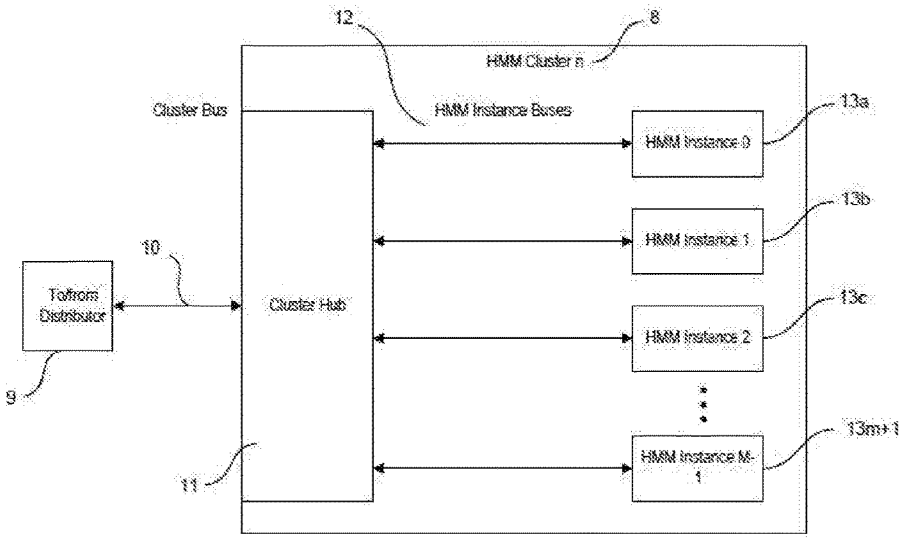

HMM Table Dimensions LOG10(freq)
Haplotype vs Read Length
7
6
5
4
3
2
1
0
Series2
Series1

FIG. 13



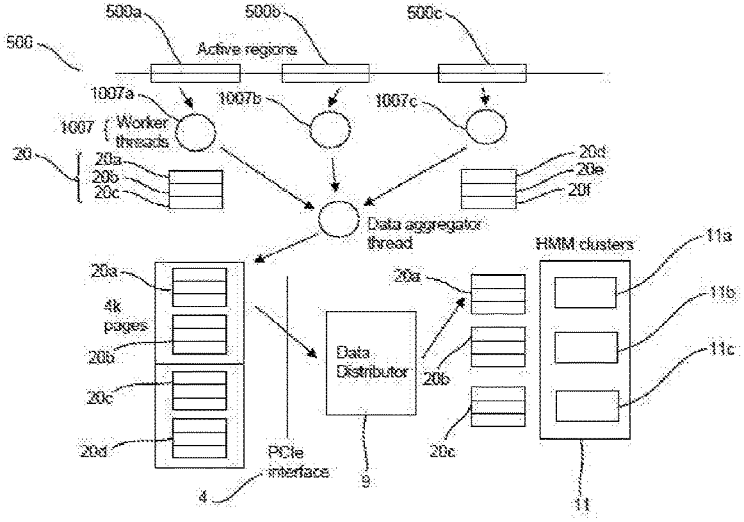
High-level view of HMM interface structure.

FIG. 14



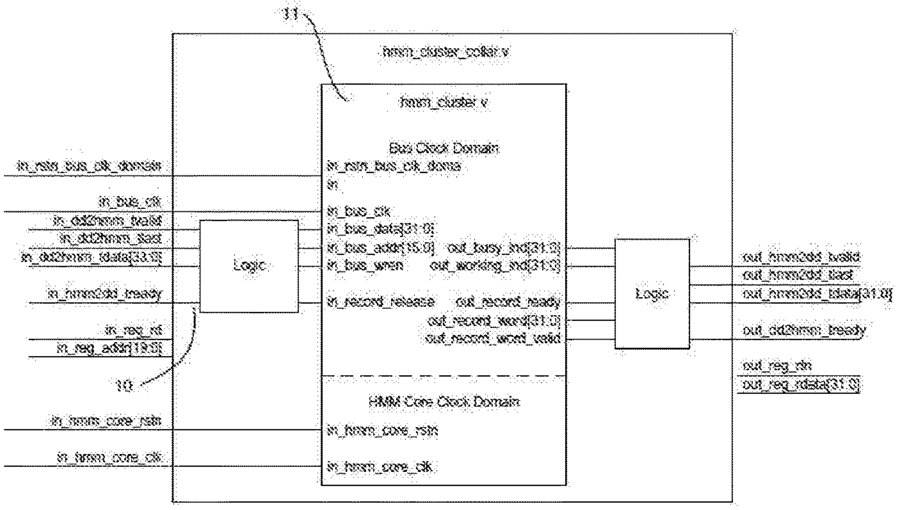
HMM cluster features.

FIG. 15



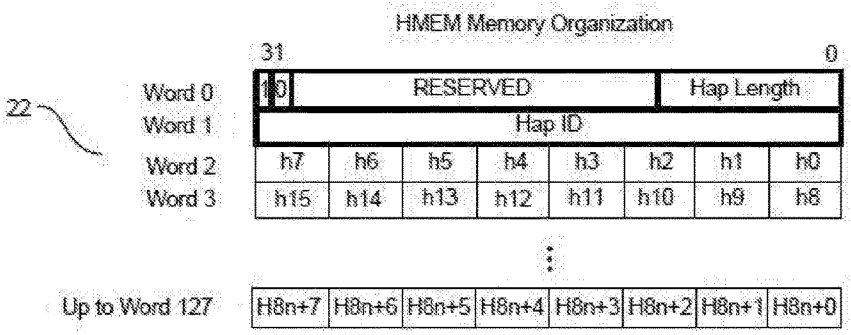
HMM data flow and HW/SW interaction overview.

FIG. 16



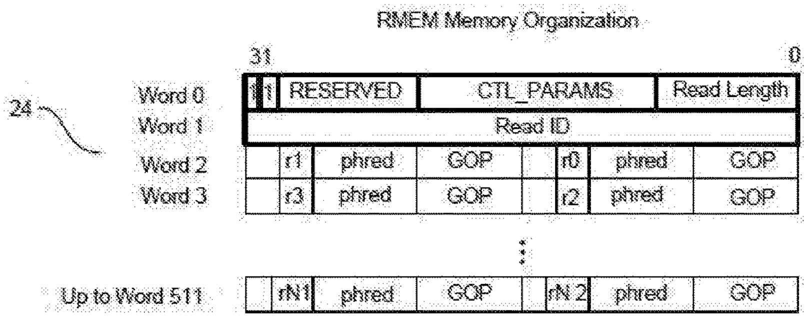
HMM Cluster Collar connections.

FIG. 17



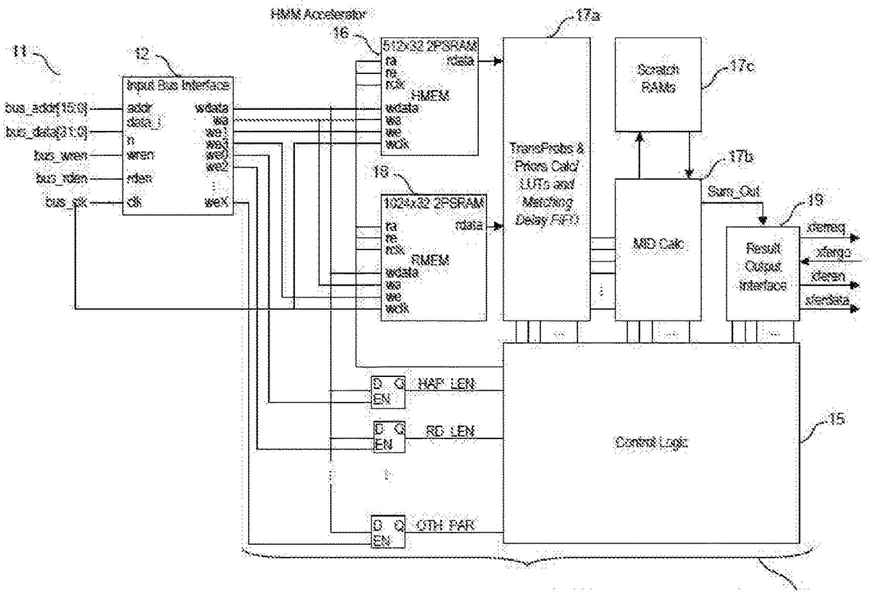
HMM Engine HMEM organization example.

FIG. 18



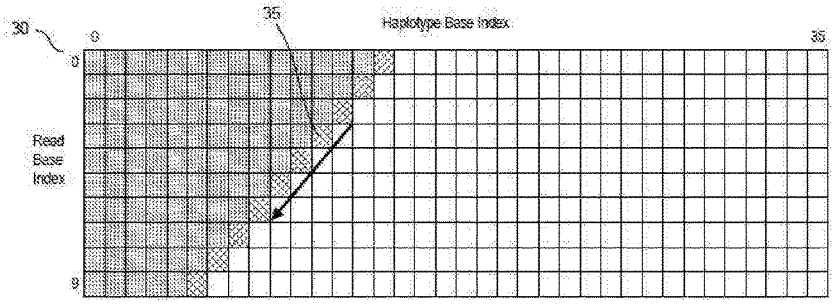
HMM RMEM organization example.

FIG. 19



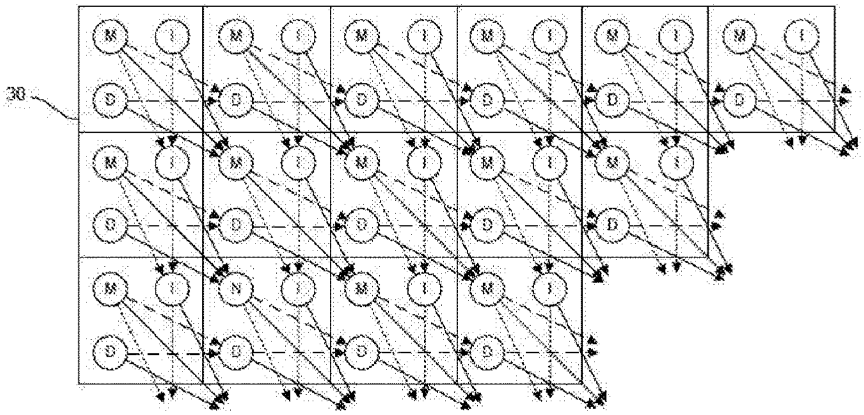
Somewhat high-level view of the major functional blocks within the HMM hardware accelerator.

FIG. 20



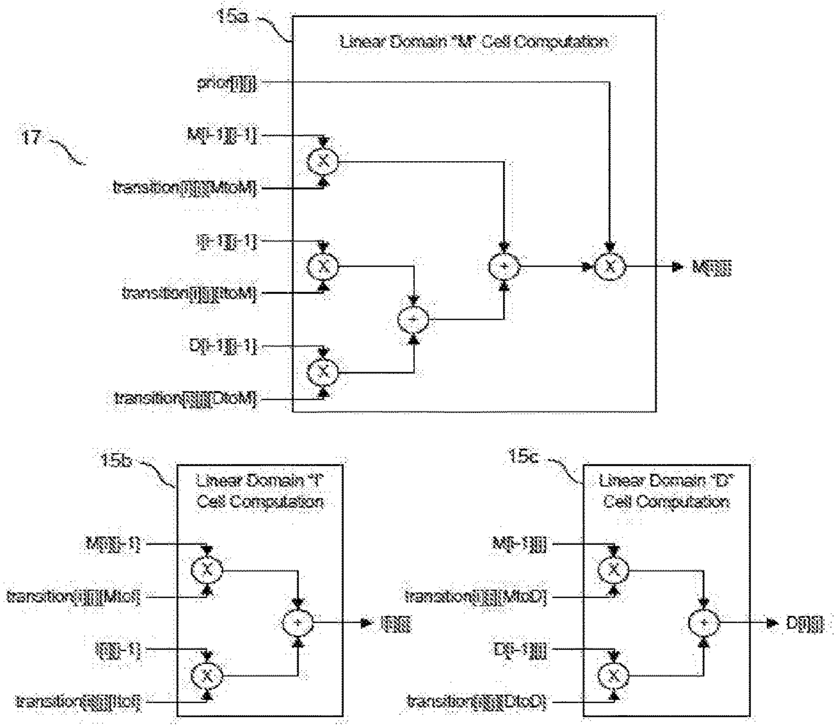
HMM matrix structure and hardware processing flow example.

FIG. 21



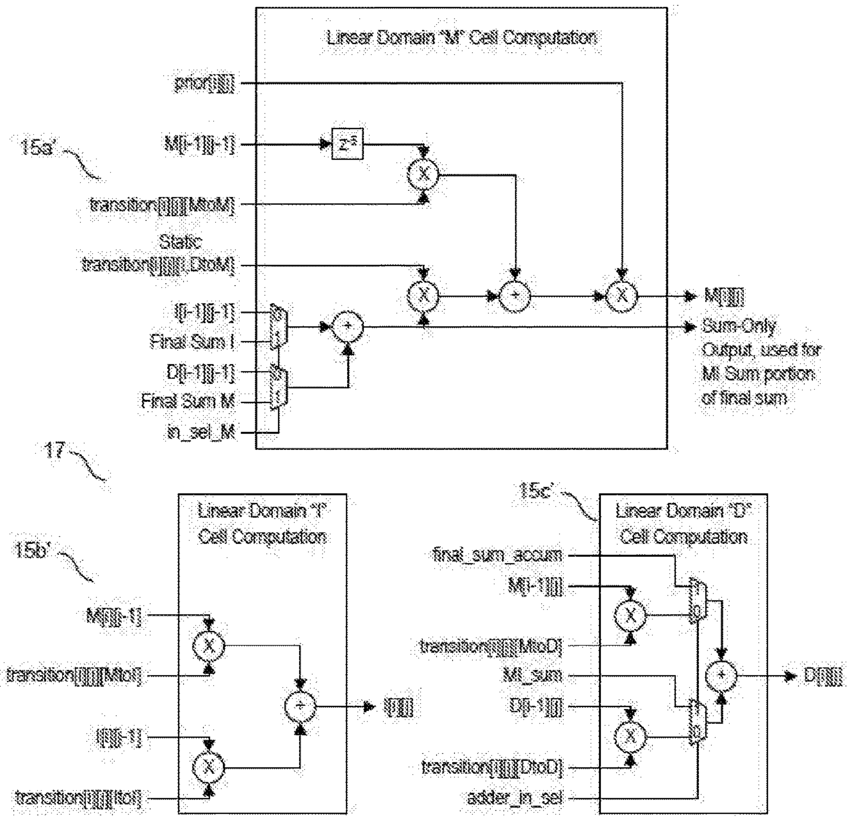
Enlarged view of a portion of Figure 8 showing the data flow and dependencies between nearby cells in the HMM M,I, and D state computations.

FIG. 22



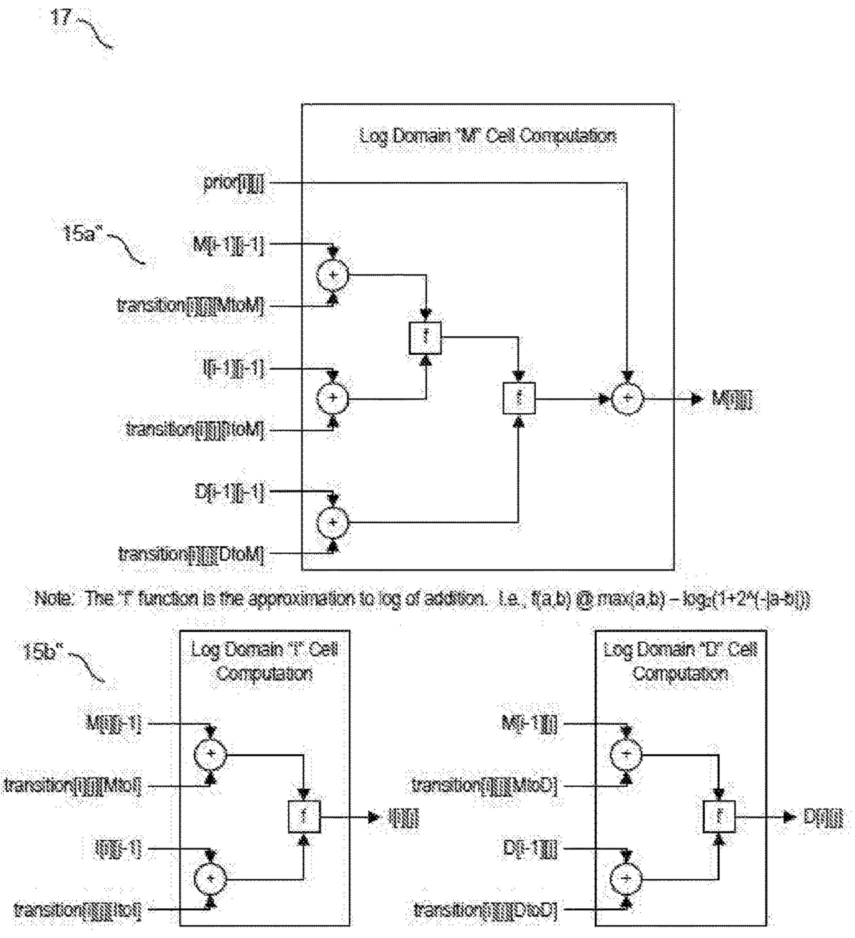
Computations required for M, I, D state updates.

FIG. 23



M,1, and D state update circuits, including effects of simplifying assumptions related to transition probabilities and the effect of sharing some M, I, D adder resources with the final sum operations.

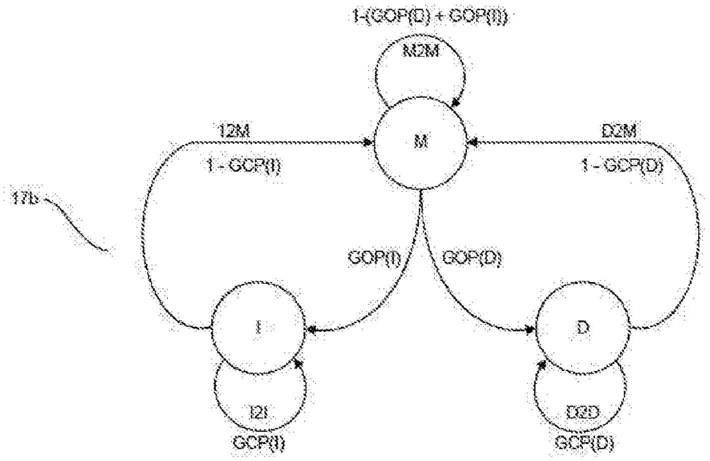
FIG. 24



Note: The "f" function is the approximation to log of addition. I.e., f(a,b) @ max(a,b) – log$_2$(1+2^(-|a-b|))

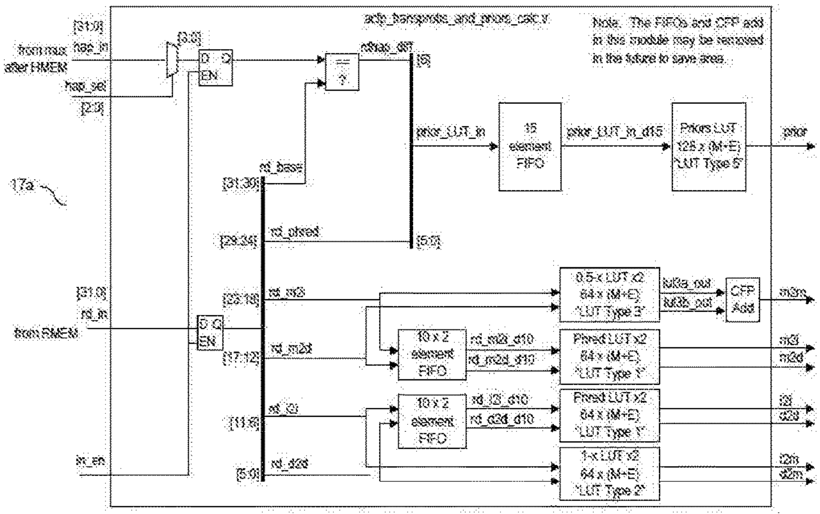
Log domain M, I, D state calculation details.

FIG. 25



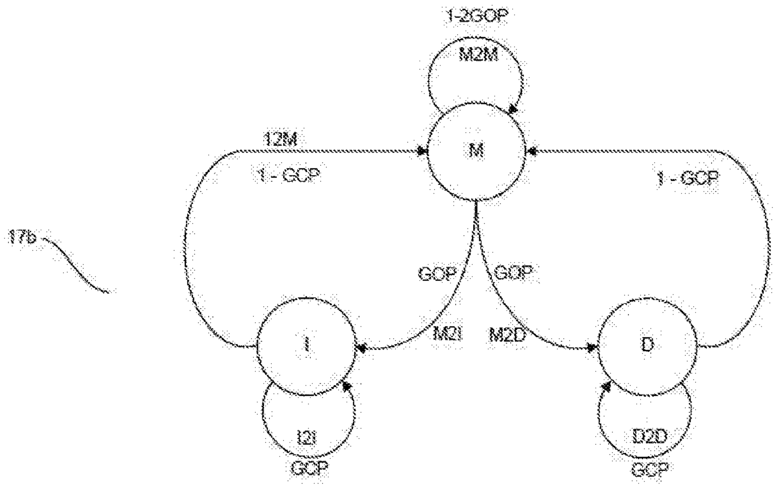
HMM state transition diagram showing relation between GOP, GCP and transition probabilities.

FIG. 26



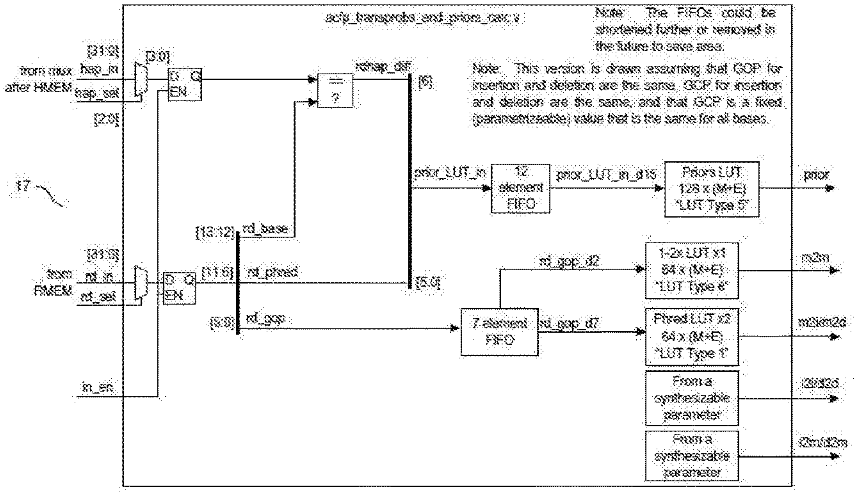
HMM Transprobs and priors generation circuit to support the general state transaction diagram of Figure 17.

FIG. 27



Simplified HMM state transition diagram showing relation between GOP, GCP and transition probabilities.

FIG. 28

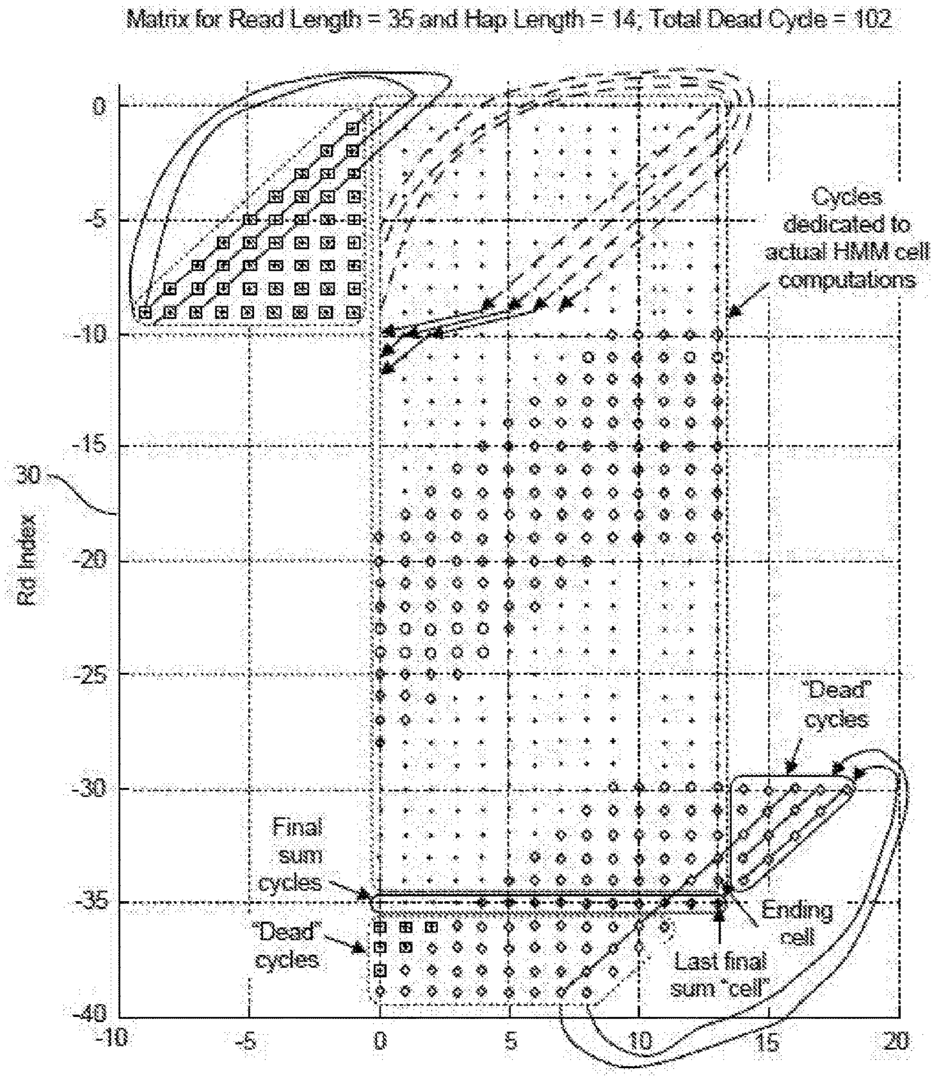


Note: The FIFOs could be shortened further or removed in the future to save area.

Note: This version is drawn assuming that GOP for insertion and deletion are the same, GCP for insertion and deletion are the same, and that GCP is a fixed (parametrizeable) value that is the same for all bases.

HMM transprobs and priors generation circuit to support the simplified state transition diagram of Figure 19.

FIG. 29


Matrix for Read Length = 35 and Hap Length = 14, Total Dead Cycle = 102
Cycles dedicated to actual HMM cell computations
"Dead" cycles
Final sum cycles
"Dead" cycles
Ending cell
Last final sum "cell"
30
Rd Index
0
-5
-10
-15
-20
-25
-30
-35
-40
-10
-5
0
5
10
15
20

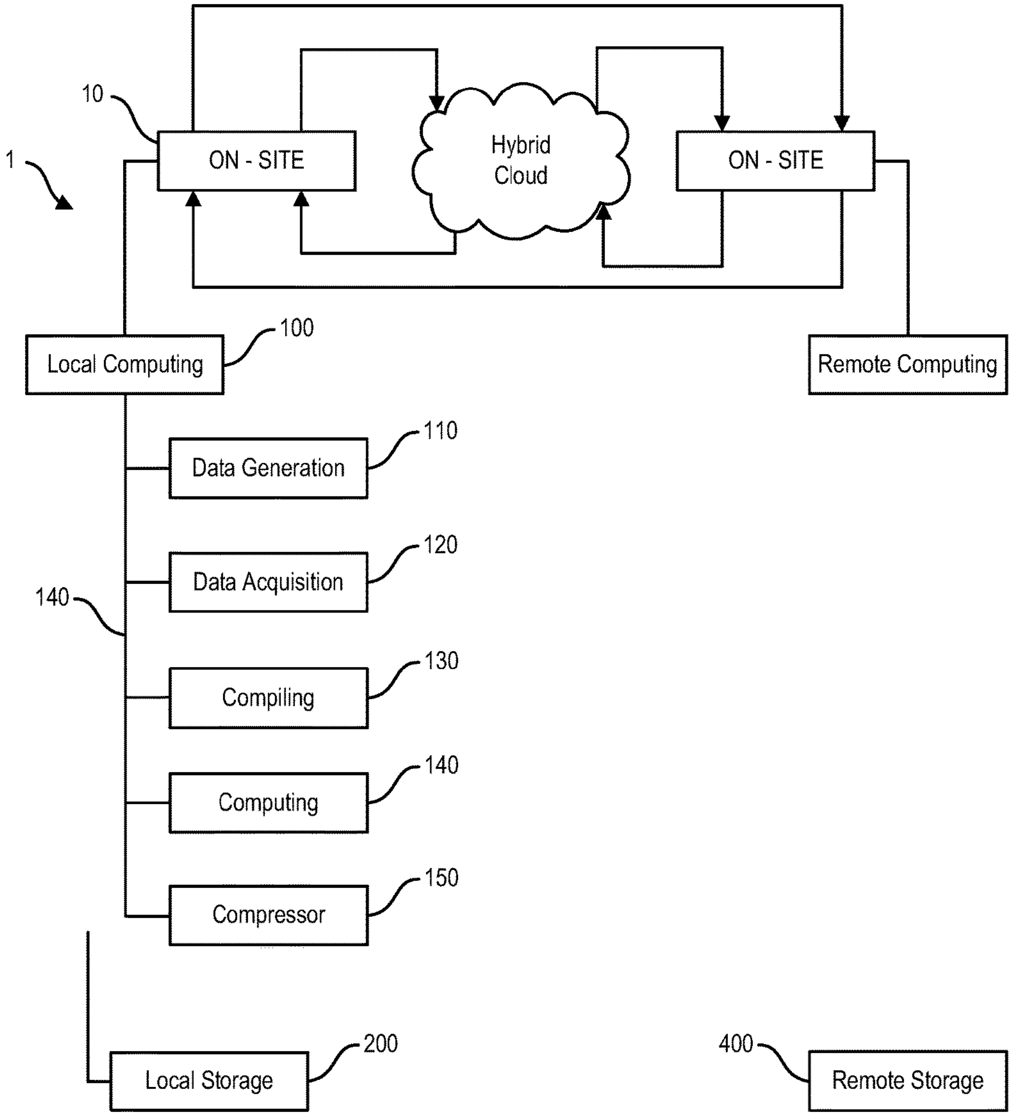

10
1
ON - SITE
Hybrid Cloud
ON - SITE
Local Computing
100
Remote Computing
Data Generation
110
Data Acquisition
120
140
Compiling
130
Computing
140
Compressor
150
Local Storage
200
400
Remote Storage

FIG. 30

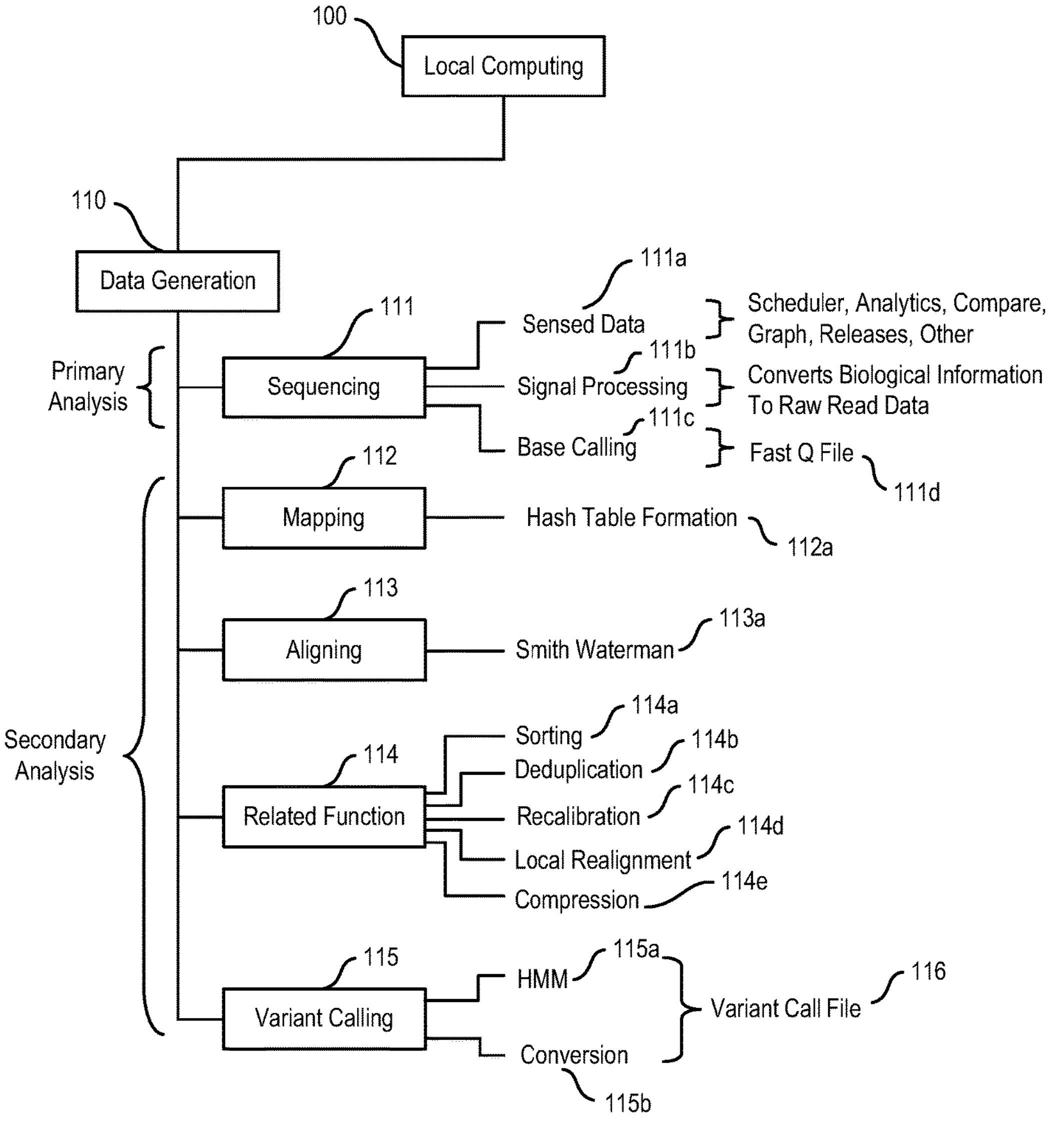

100
Local Computing
110
Data Generation
Primary Analysis
111
Sequencing
111a
Sensed Data
Scheduler, Analytics, Compare, Graph, Releases, Other
111b
Signal Processing
Converts Biological Information To Raw Read Data
111c
Base Calling
Fast Q File
111d
Secondary Analysis
112
Mapping
Hash Table Formation
112a
113
Aligning
Smith Waterman
113a
114
Related Function
114a
Sorting
114b
Deduplication
114c
Recalibration
114d
Local Realignment
114e
Compression
115
Variant Calling
115a
HMM
Conversion
115b
Variant Call File
116

FIG. 31A

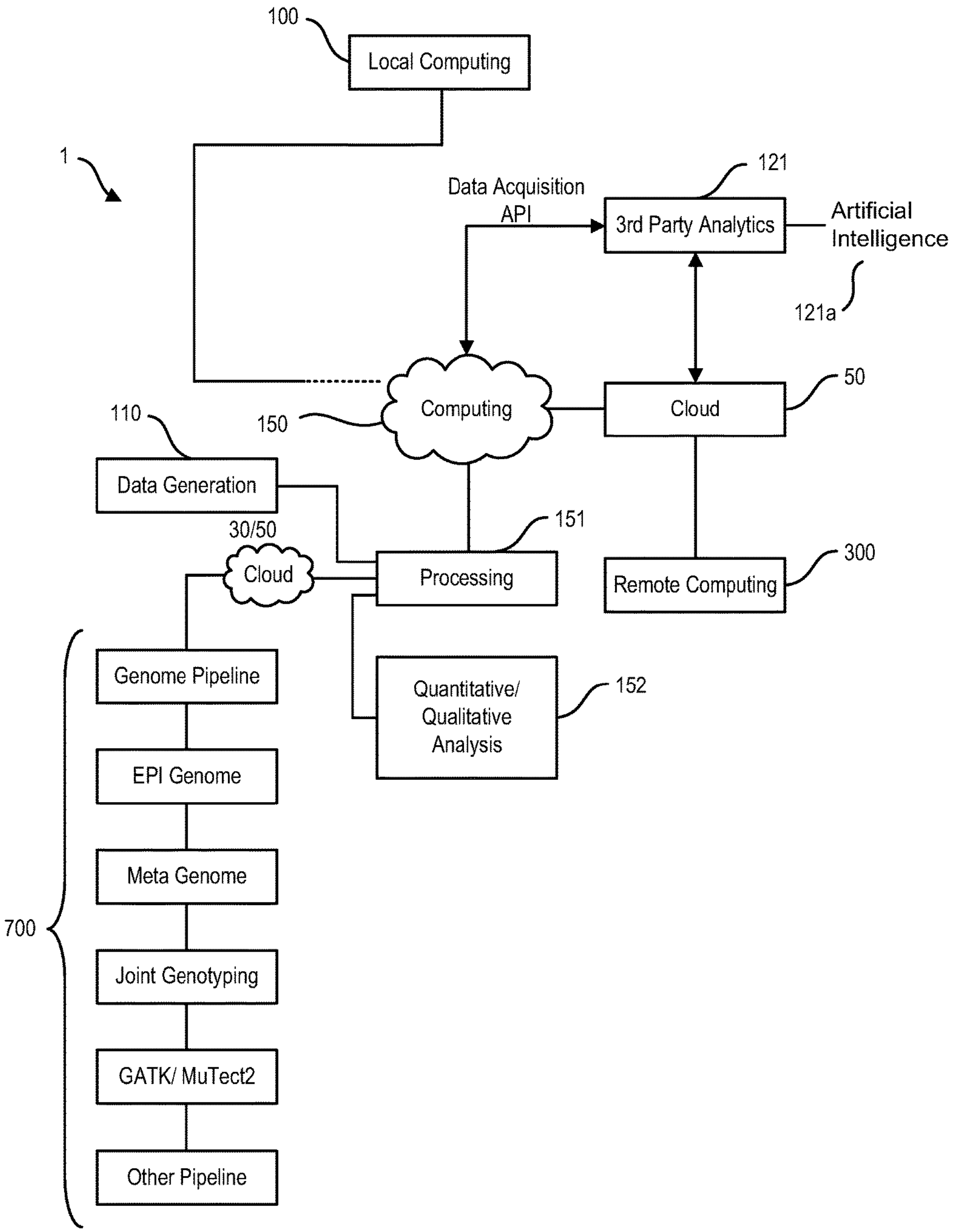

100
Local Computing
1
121
Data Acquisition API
3rd Party Analytics
Artificial Intelligence
121a
50
Computing
Cloud
110
150
Data Generation
151
30/50
Cloud
Processing
300
Remote Computing
Genome Pipeline
Quantitative/ Qualitative Analysis
152
EPI Genome
Meta Genome
700
Joint Genotyping
GATK/ MuTect2
Other Pipeline

FIG. 31B

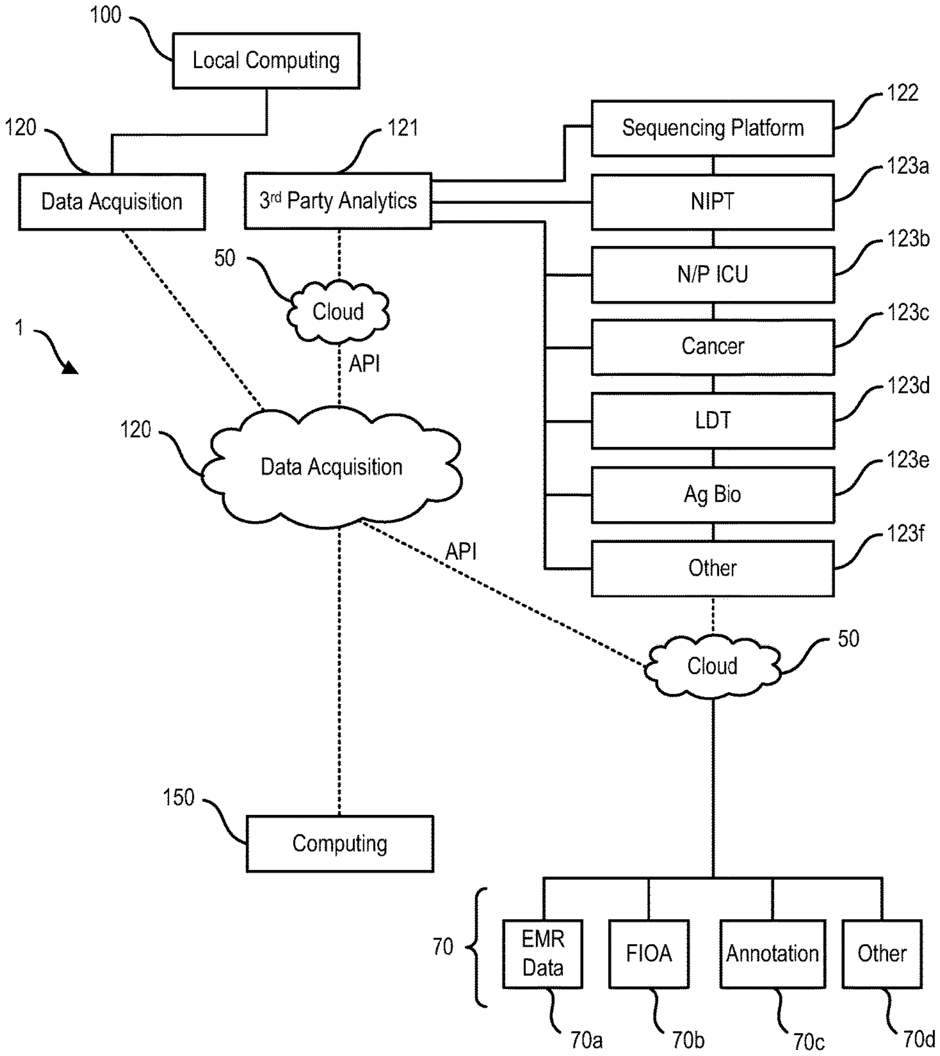

100
Local Computing
120
Data Acquisition
121
3rd Party Analytics
122
Sequencing Platform
123a
NIPT
123b
N/P ICU
123c
Cancer
123d
LDT
123e
Ag Bio
123f
Other
50
Cloud
API
1
120
Data Acquisition
API
50
Cloud
150
Computing
70
EMR Data
FIOA
Annotation
Other
70a
70b
70c
70d

FIG. 31C

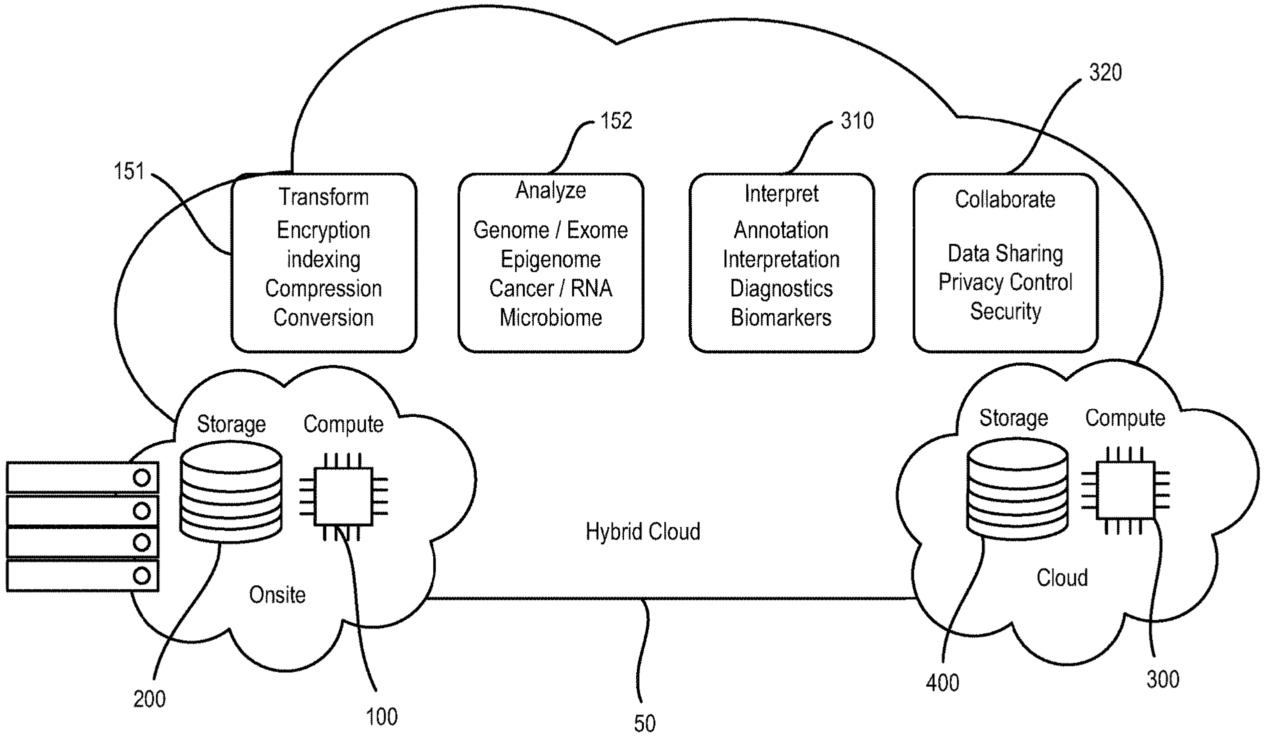

151
Transform
Encryption
indexing
Compression
Conversion
152
Analyze
Genome / Exome
Epigenome
Cancer / RNA
Microbiome
310
Interpret
Annotation
Interpretation
Diagnostics
Biomarkers
320
Collaborate
Data Sharing
Privacy Control
Security
Storage
Compute
Onsite
200
100
Hybrid Cloud
50
Storage
Compute
Cloud
400
300

FIG. 32A

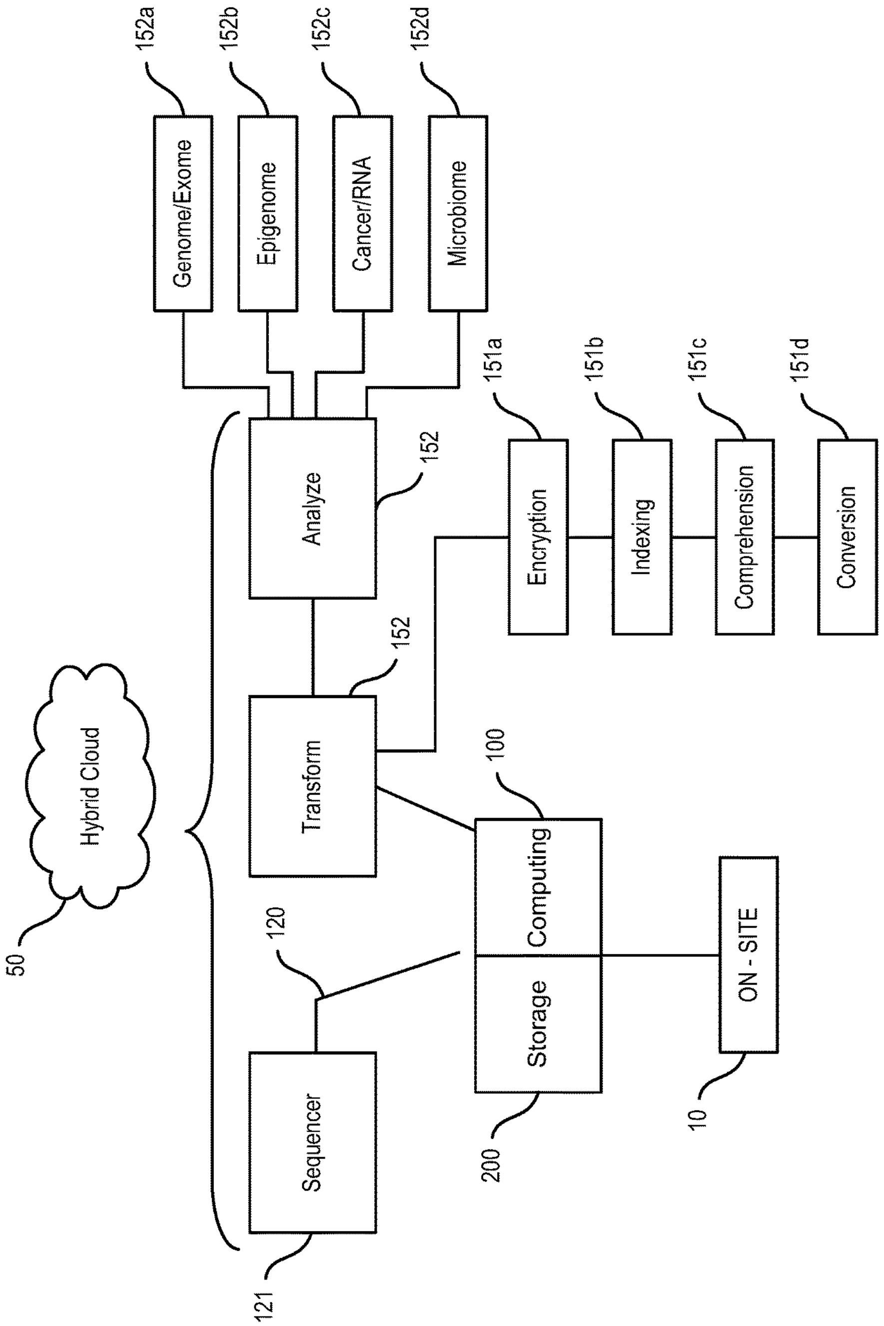

Hybrid Cloud
50
Sequencer
121
120
Storage
Computing
200
100
ON - SITE
10
Transform
152
Analyze
152
Genome/Exome
152a
Epigenome
152b
Cancer/RNA
152c
Microbiome
152d
Encryption
151a
Indexing
151b
Comprehension
151c
Conversion
151d

FIG. 32B

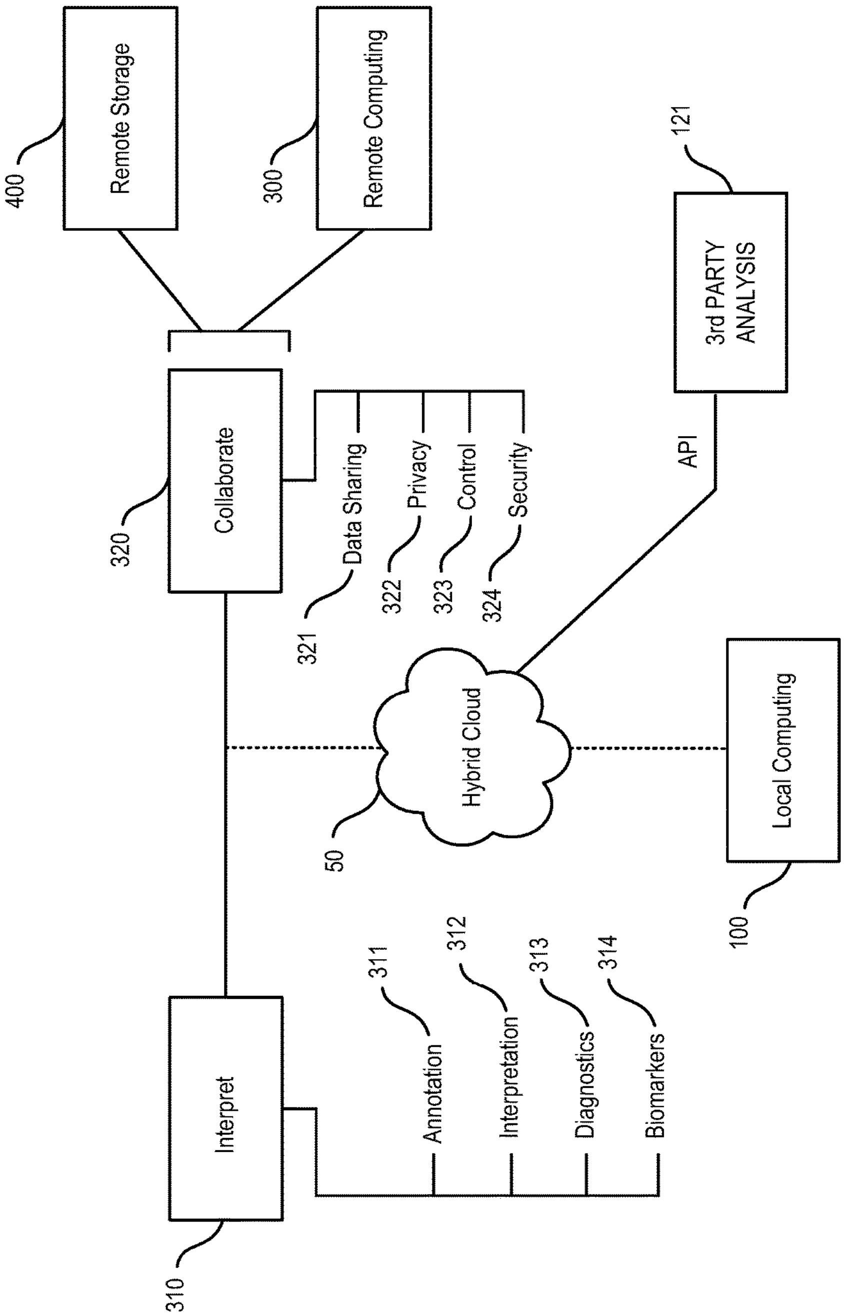

400
Remote Storage
300
Remote Computing
320
Collaborate
321
Data Sharing
322
Privacy
323
Control
324
Security
121
3rd PARTY ANALYSIS
API
50
Hybrid Cloud
100
Local Computing
310
Interpret
311
Annotation
312
Interpretation
313
Diagnostics
314
Biomarkers

FIG. 32C

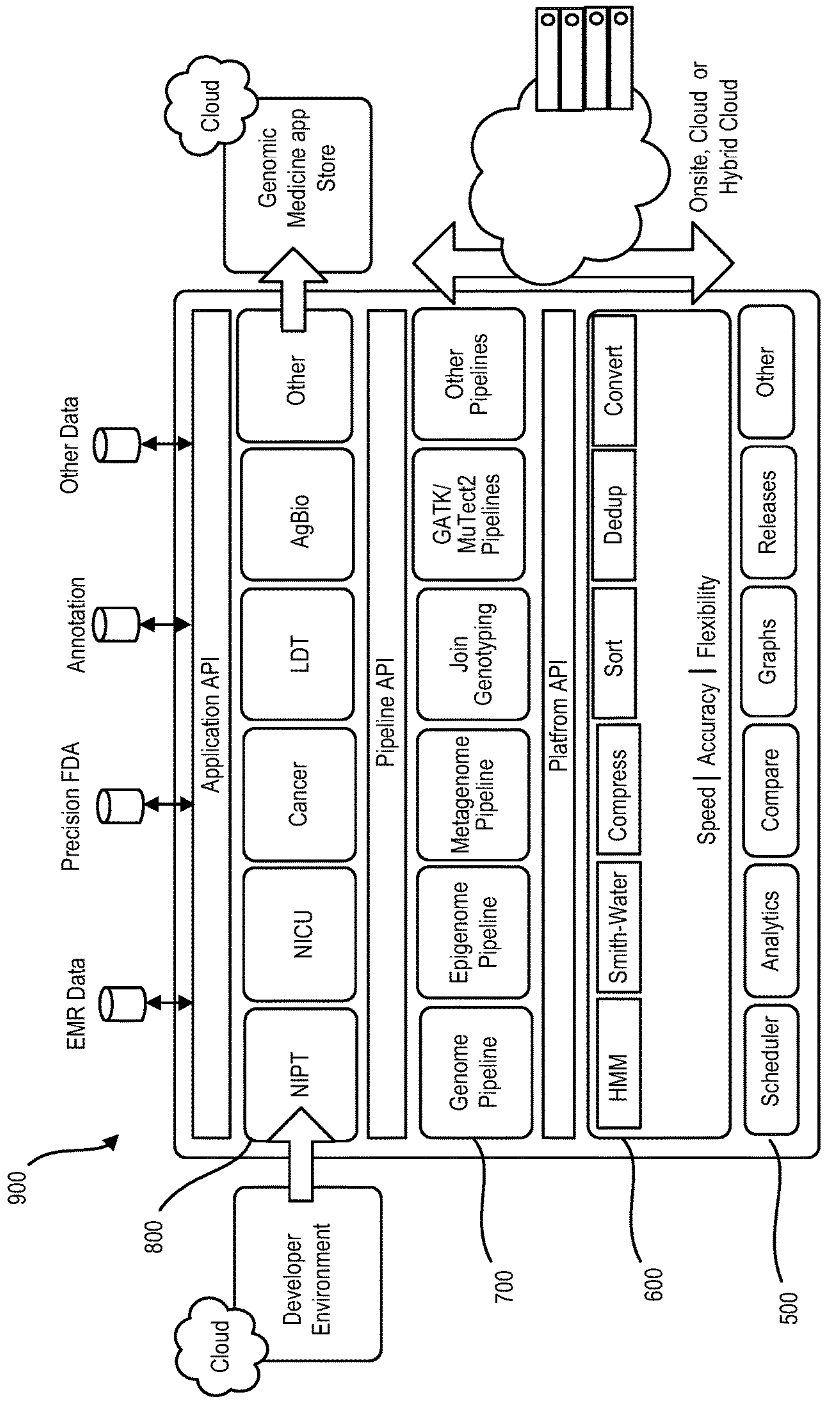

900
EMR Data
Precision FDA
Annotation
Other Data
Application API
NIPT
NICU
Cancer
LDT
AgBio
Other
Cloud
Genomic Medicine app Store
Developer Environment
Cloud
800
Pipeline API
Genome Pipeline
Epigenome Pipeline
Metagenome Pipeline
Join Genotyping
GATK/ MuTect2 Pipelines
Other Pipelines
700
Platfrom API
HMM
Smith-Water
Compress
Sort
Dedup
Convert
Speed | Accuracy | Flexibility
600
Scheduler
Analytics
Compare
Graphs
Releases
Other
500
Onsite, Cloud or Hybrid Cloud

FIG. 33

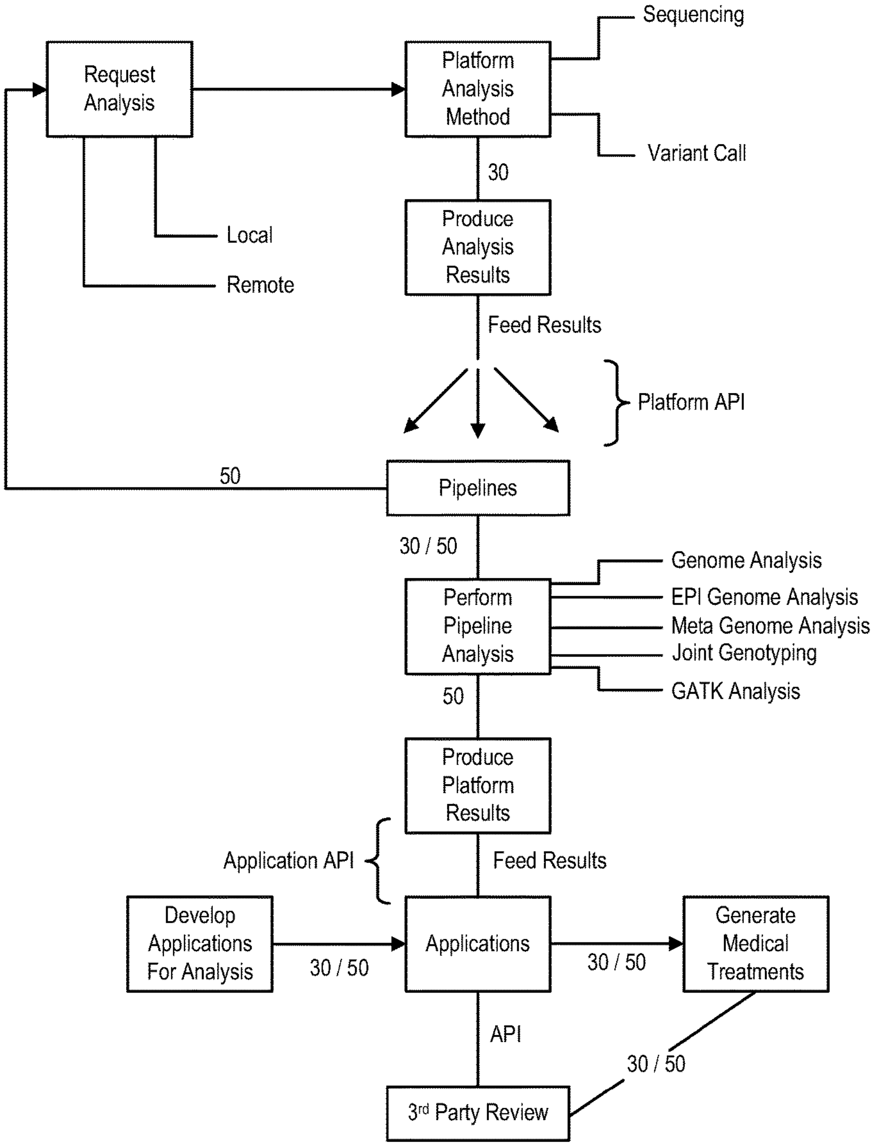

Request Analysis
Platform Analysis Method
Sequencing
Variant Call
30
Local
Remote
Produce Analysis Results
Feed Results
Platform API
50
Pipelines
30 / 50
Genome Analysis
EPI Genome Analysis
Meta Genome Analysis
Joint Genotyping
GATK Analysis
Perform Pipeline Analysis
50
Produce Platform Results
Application API
Feed Results
Develop Applications For Analysis
30 / 50
Applications
30 / 50
Generate Medical Treatments
API
30 / 50
3rd Party Review

FIG. 34

FIG. 35

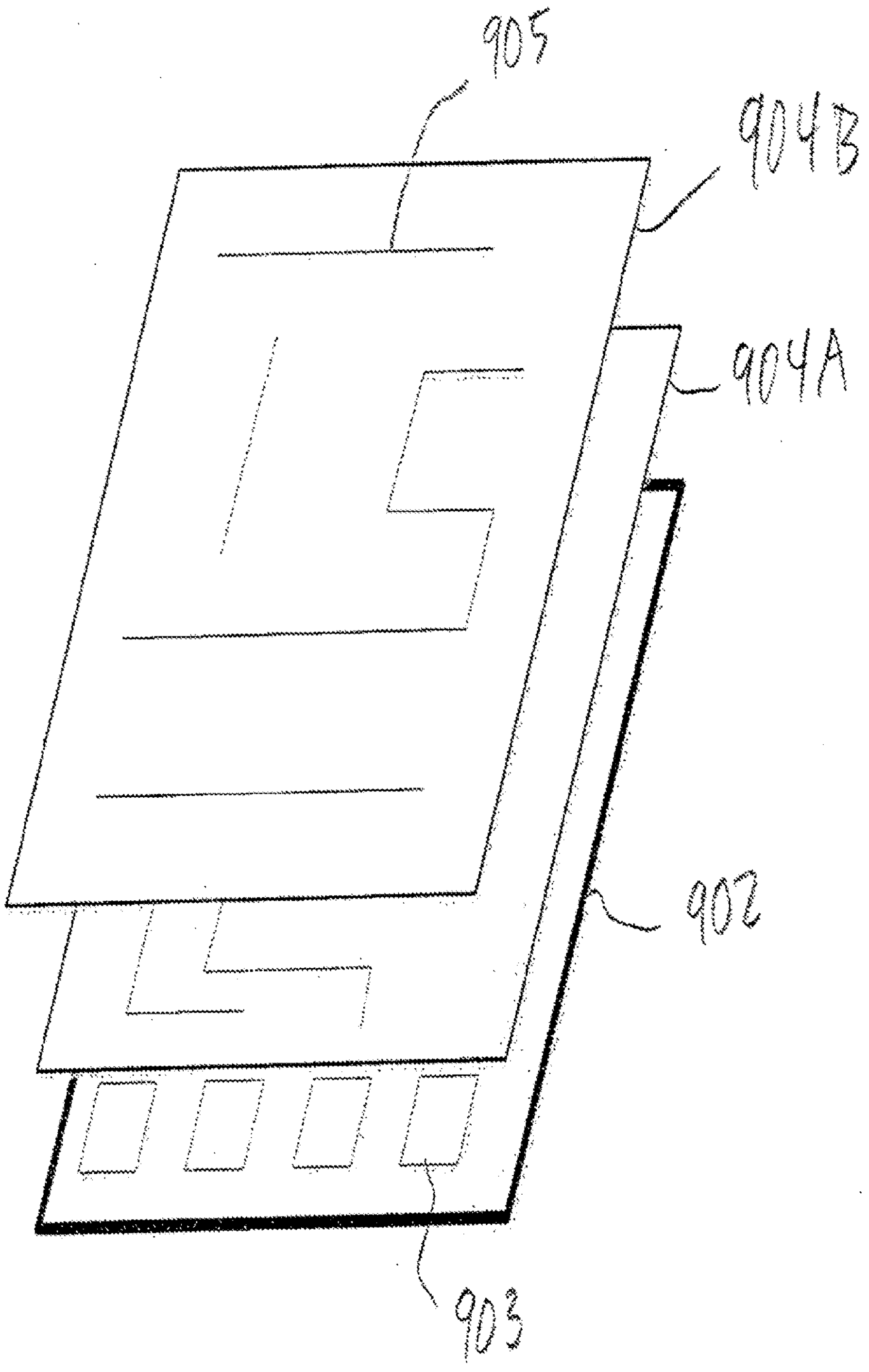


FIG. 36

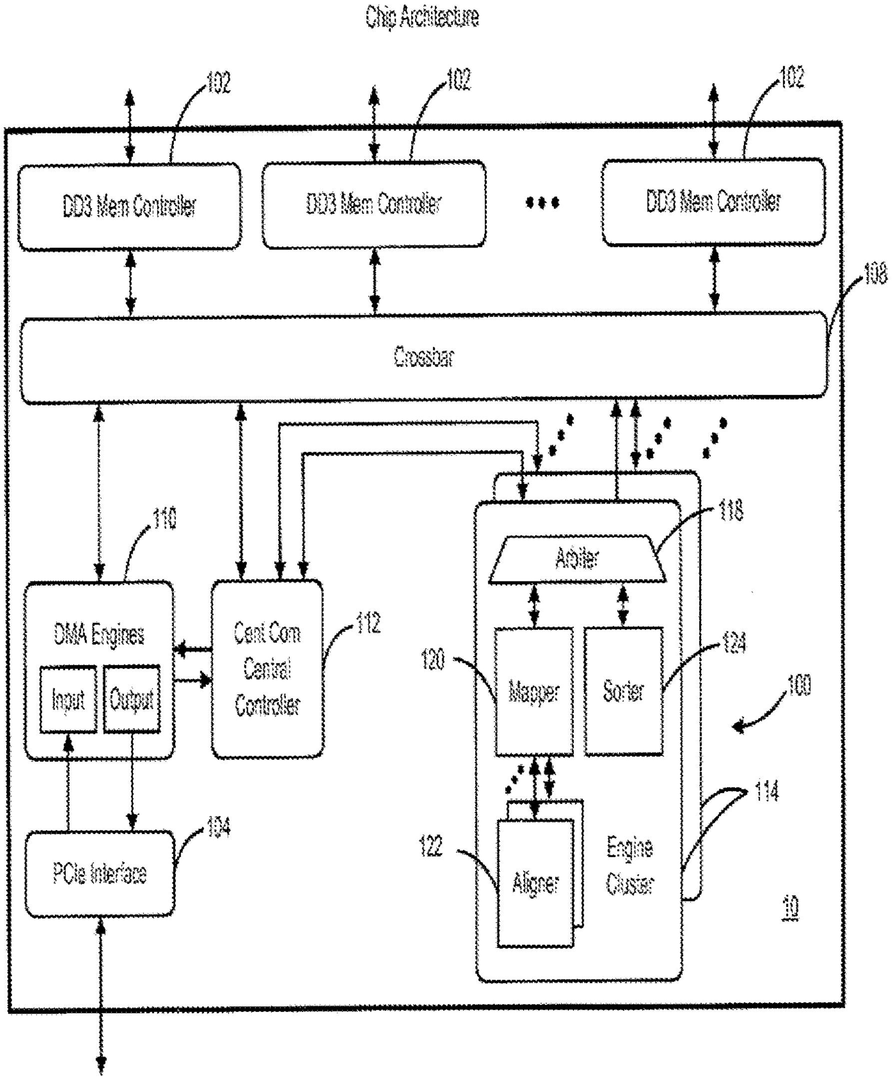

Chip Architecture
102
102
102
DD3 Mem Controller
DD3 Mem Controller
DD3 Mem Controller
108
Crossbar
110
DMA Engines
Input
Output
Cent Com Central Controller
112
104
PCIe Interface
118
Arbiter
120
Mapper
Sorter
124
100
122
Aligner
Engine Cluster
114
10

FIG. 37

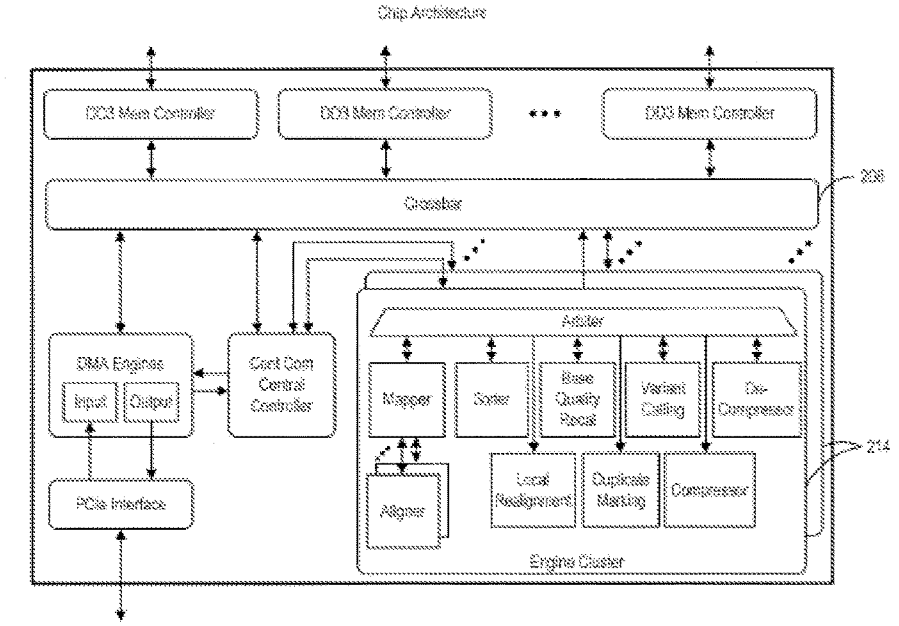

Chip Architecture
DD3 Mem Controller
DD3 Mem Controller
DD3 Mem Controller
208
Crossbar
DMA Engines
Input
Output
PCIe Interface
Card Com Central Controller
Arbiter
Mapper
Sorter
Base Quality Recal
Variant Calling
De-Compressor
Aligner
Local Realignment
Duplicate Marking
Compressor
214
Engine Cluster

FIG. 38

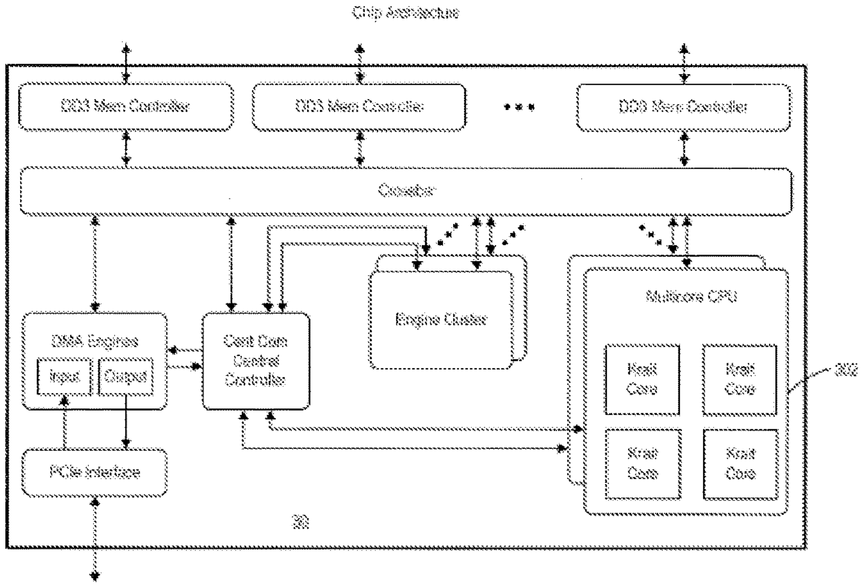

Chip Architecture
DD3 Mem Controller
DD3 Mem Controller
DD3 Mem Controller
Crossbar
DMA Engines
Input
Output
PCIe Interface
Card Com Central Controller
Engine Cluster
Multicore CPU
Krait Core
Krait Core
Krait Core
Krait Core
302
30

FIG. 39

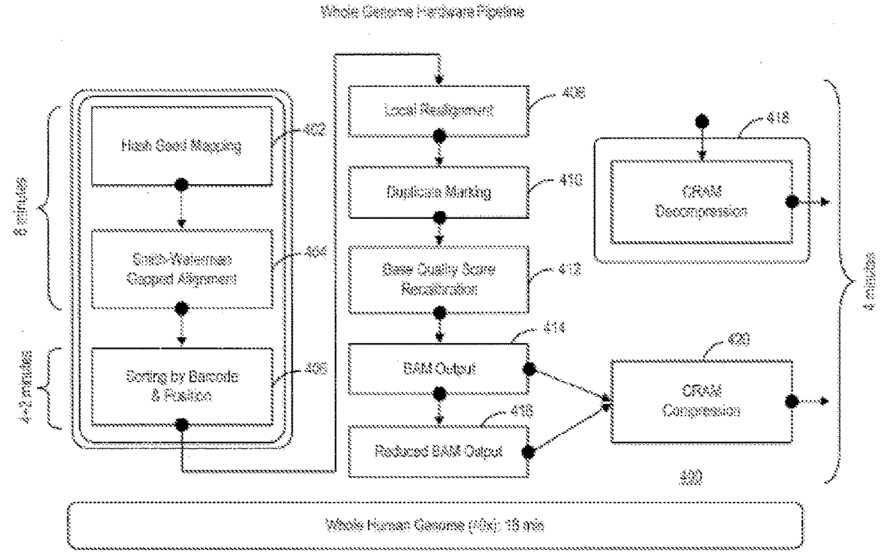

Whole Genome Hardware Pipeline
8 minutes
1-2 minutes
Hash Seed Mapping
402
Smith-Waterman Gapped Alignment
404
Sorting by Barcode & Position
406
Local Realignment
408
Duplicate Marking
410
Base Quality Score Recalibration
412
414
BAM Output
416
Reduced BAM Output
418
CRAM Decompression
420
CRAM Compression
400
4 minutes
Whole Human Genome (40x): 15 min

FIG. 40

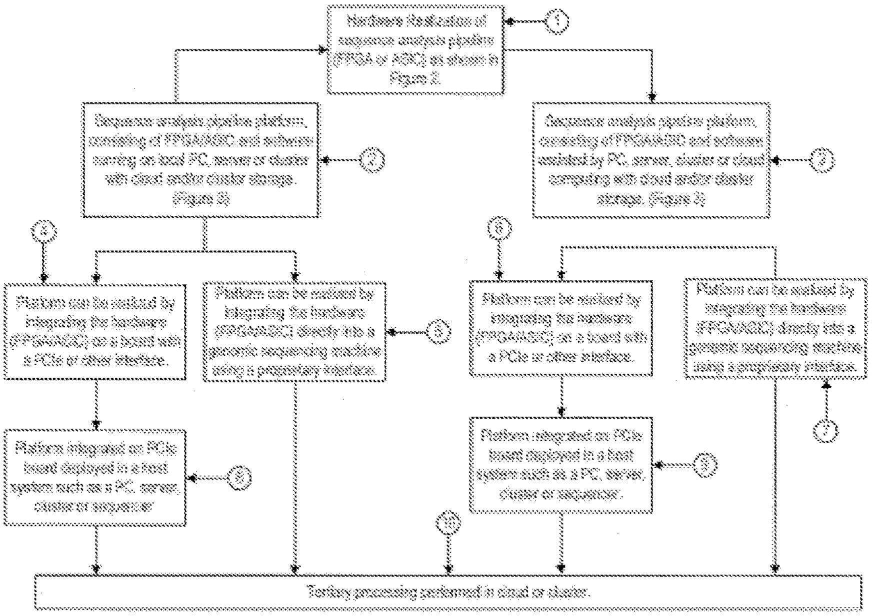

Hardware Realization of sequence analysis pipeline (FPGA or ASIC) as shown in Figure 2.
1
Sequence analysis pipeline platform, consisting of FPGA/ASIC and software running on local PC, server or cluster with cloud and/or cluster storage. (Figure 3)
2
Sequence analysis pipeline platform, consisting of FPGA/ASIC and software assisted by PC, server, cluster or cloud computing with cloud and/or cluster storage. (Figure 3)
3
4
Platform can be realized by integrating the hardware (FPGA/ASIC) on a board with a PCIe or other interface.
Platform can be realized by integrating the hardware (FPGA/ASIC) directly into a genomic sequencing machine using a proprietary interface.
5
6
Platform can be realized by integrating the hardware (FPGA/ASIC) on a board with a PCIe or other interface.
Platform can be realized by integrating the hardware (FPGA/ASIC) directly into a genomic sequencing machine using a proprietary interface.
7
Platform integrated on PCIe board deployed in a host system such as a PC, server, cluster or sequencer.
8
Platform integrated on PCIe board deployed in a host system such as a PC, server, cluster or sequencer.
9
10
Tertiary processing performed in cloud or cluster.

FIG. 41

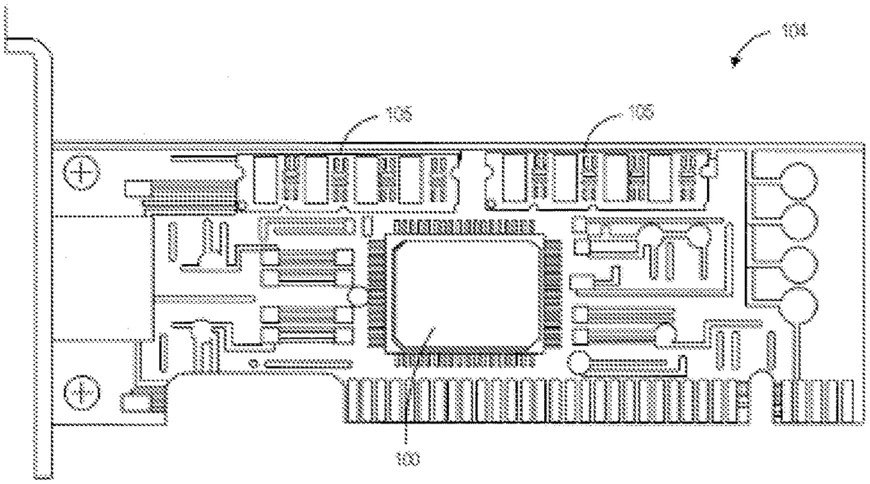


FIG. 42

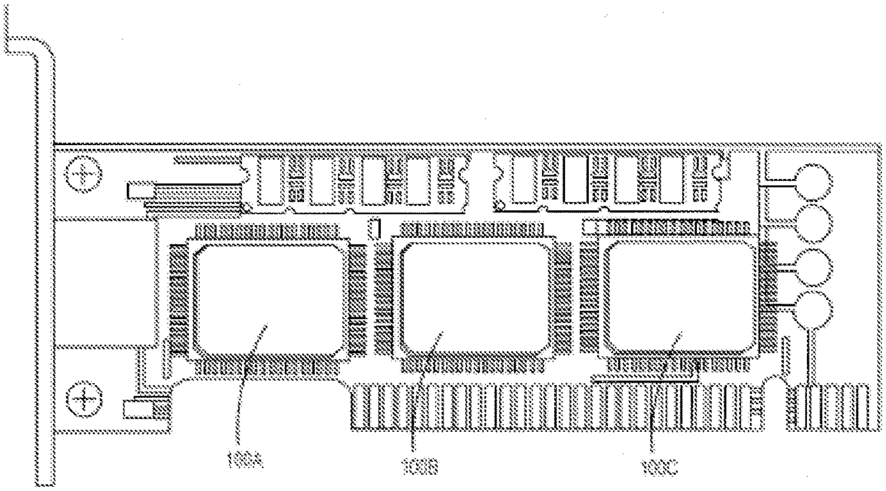
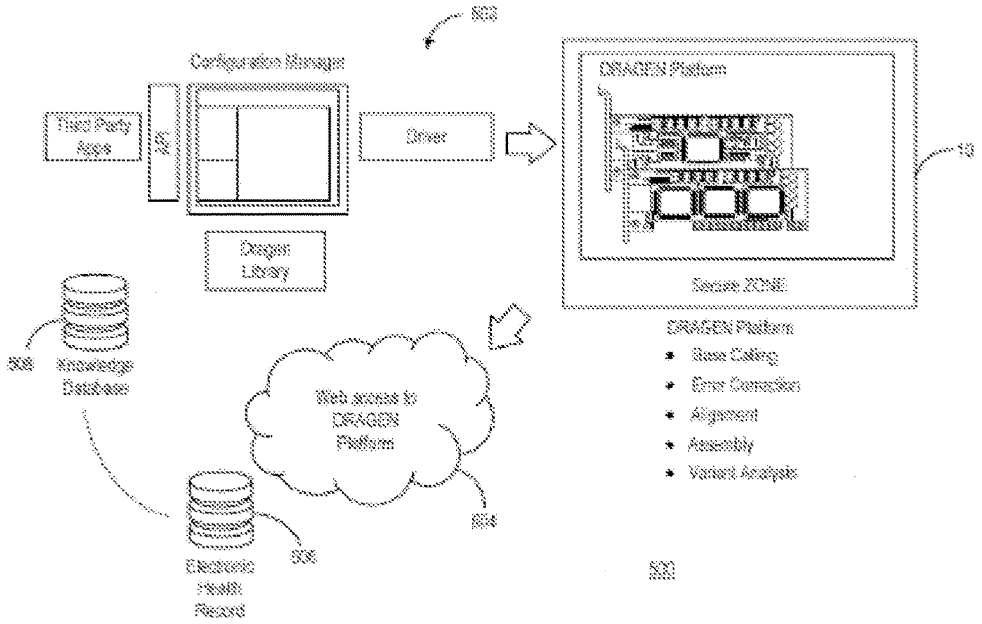


FIG. 43


802
Configuration Manager
Third Party Apps
API
Driver
Dragen Library
DRAGEN Platform
10
Secure ZONE
DRAGEN Platform
* Base Calling
* Error Correction
* Alignment
* Assembly
* Variant Analysis
808
Knowledge Database
Web access to DRAGEN Platform
804
806
Electronic Health Record
800

BIOINFORMATICS SYSTEMS, APPARATUSES, AND METHODS EXECUTED ON AN INTEGRATED CIRCUIT PROCESSING PLATFORM

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application is a continuation of U.S. patent application Ser. No. 15/059,221, filed Mar. 2, 2016, which claims the benefit of U.S. Provisional Application Ser. No. 62/127,232, filed Mar. 2, 2015.

[0002] U.S. patent application Ser. No. 15/059,221, filed Mar. 2, 2016, is also a continuation in part of U.S. patent application Ser. No. 15/048,935, filed Feb. 19, 2016, which claims the benefit of U.S. Provisional Application Ser. No. 62/119,059, filed on Feb. 20, 2015 and U.S. Provisional Application Ser. No. 62/127,232, filed on Mar. 2, 2015.

[0003] U.S. patent application Ser. No. 15/048,935, filed Feb. 19, 2016, is a continuation in part of U.S. patent application Ser. No. 14/284,307, filed May 21, 2014, now U.S. Pat. No. 9,235,680, which is a continuation of U.S. patent application Ser. No. 14/279,063, filed May 15, 2014, now U.S. Pat. No. 9,679,104, which claims the benefit of U.S. Provisional Application Ser. No. 61/988,128 filed May 2, 2014, U.S. Provisional Application Ser. No. 61/984,663 filed Apr. 25, 2014, and U.S. Provisional Application Ser. No. 61/943,870 filed Feb. 24, 2014.

[0004] U.S. patent application Ser. No. 14/279,063 is also a continuation-in-part of patent application Ser. No. 14/158,758, filed Jan. 17, 2014, now U.S. Pat. No. 9,483,610, which claims the benefit of U.S. Provisional Application Ser. No. 61/910,868, filed Dec. 2, 2013, U.S. Provisional Application Ser. No. 61/823,824, filed May 15, 2013, U.S. Provisional Application Ser. No. 61/822,101, filed May 10, 2013, U.S. Provisional Application Ser. No. 61/826,381 filed May 22, 2013, and U.S. Provisional Application Ser. No. 61/753,775 field Jan. 17, 2013.

[0005] The disclosures of the above-identified patent applications are hereby incorporated by reference in their entirety.

TECHNICAL FIELD

[0006] The subject matter described herein relates to bioinformatics, and more particularly to systems, apparatuses, and methods for implementing bioinformatic protocols, such as performing one or more functions for analyzing genomic data on an integrated circuit, such as on a hardware processing platform.

BACKGROUND

[0007] A goal for health care researchers and practitioners is to improve the safety, quality, and effectiveness of health care for every patient. Personalized health care is directed to achieving these goals on an individual level. For instance, "genomics" and/or "bioinformatics" are fields of study that aim to facilitate the safety, the quality, and the effectiveness of prophylactic and therapeutic treatments on a personalized, individual level. Accordingly, by employing genomics and/or bioinformatics techniques, the identity of an individual's genetic makeup, e.g., his or hers genes, may be determined and that knowledge may be used in the development of therapeutic and/or prophylactic regimens, including drug treatments, that are personalized to the individual, thus, enabling medicine to be tailored to meet each person's individual needs.

[0008] The desire to provide personalized care to individuals is transforming the health care system. This transformation of the health care system is likely to be powered by breakthrough innovations at the intersection of medical science and information technology such as is represented by the fields of genomics and bioinformatics. Accordingly, genomics and bioinformatics are key foundations upon which this future will be built. Science has evolved dramatically since the first human genome was fully sequenced in 2000 at a total cost of over $1 Billion. Today, we are on the verge of high resolution sequencing at a cost of less than $1K per genome, making it economically feasible for the first time to move out of the research lab and into widespread adoption for medical care. Genomic data, therefore, may become a vital input to diagnostic screening, therapeutic and/or prophylactic drug discovery, and/or disease treatment.

[0009] More particularly, genomics and bioinformatics are fields concerned with the application of information technology and computer science to the field of molecular biology. In particular, bioinformatics techniques can be applied to process and analyze various genomic data, such as from an individual so as to determine qualitative and quantitative information about that data that can then be used by various practitioners in the development of prophylactic and therapeutic methods for preventing or at least ameliorating diseased states, and thus, improving the safety, quality, and effectiveness of health care on an individualized level.

[0010] Because of its focus on advancing personalized healthcare, bioinformatics, therefore, promotes individualized healthcare that is proactive, instead of reactive, and this gives the patient the opportunity to become more involved in their own wellness. Typically, this can be achieved through two guiding principles. First, federal leadership can be provided to support research that addresses these individual aspects of disease and disease prevention, such as with the ultimate goal of shaping diagnostic and preventative care to match each person's unique genetic characteristics. Additionally, a "network of networks" may be created to aggregate health care data to help researchers establish patterns and identify genetic "definitions" to existing diseases.

[0011] An advantage of employing bioinformatics technologies in such instances is that the qualitative and/or quantitative analyses of molecular biological data can be performed on a broader range of sample sets at a much higher rate of speed and often times more accurately, thus expediting the emergence of a personalized healthcare system.

[0012] Accordingly, in various instances, the molecular data to be processed in a bioinformatics based platform typically concerns genomic data, such as Deoxyribonucleic acid (DNA) and/or Ribonucleic acid (RNA) data. For example, a well-known method for generating DNA and/or RNA data involves DNA/RNA sequencing. DNA/RNA sequencing can be performed manually, such as in a lab, or may be performed by an automated sequencer, such as at a core sequencing facility, for the purpose of determining the genetic makeup of a sample of an individual's genetic material, e.g., DNA and/or RNA. The person's genetic information may then be used in comparison to a referent, such as a reference sequence, haplotype, or theoretical haplotype, so as to determine its variance therefrom. Such variant information may then be subjected to further processing and used to determine or predict the occurrence of a diseased state in the individual.

[0013] For instance, manual or automated DNA/RNA sequencing may be employed to determine the sequence of nucleotide bases in a sample of DNA/RNA, such as a sample obtained from a subject. Using various different genomics techniques these sequences may then be strung together to generate the genomic sequence of the subject. This sequence may then be compared to a reference genomic sequence to determine how the genomic sequence of the subject varies from that of the reference. Such a process involves determining the variants in the sampled sequence and presents a central challenge to genomics and bioinformatics methodologies.

[0014] For example, a central challenge in DNA sequencing is building full-length genomic sequences, e.g., chromosomal sequences, from a sample of genetic material that can be compared to a reference genomic sequence such as to determine the variants in the sampled full-length genomic sequences. In particular, the methods employed in sequencing protocols do not produce full-length chromosomal sequences of the sample DNA.

[0015] Rather, sequence fragments, typically from 100-1,000 nucleotides in length, are produced without any indication as to where in the genome they align. Therefore, in order to generate full length chromosomal genomic constructs, these fragments of DNA sequences, called "reads" need to be mapped, aligned, merged, sorted, and/or compared to a reference genomic sequence. Through such processes the variants of the sample genomic sequences from the reference genomic sequences may be determined.

[0016] However, as the human genome is comprised of approximately 3.1 billion base pairs, and as each sequence fragment, or read, is typically only from 100 to 500 or 1000 nucleotides in length, the time and effort that goes into building such full length genomic sequences and determining the variants therein is quite extensive often requiring the use of several different computer resources applying several different algorithms over prolonged periods of time.

[0017] In a particular instance, thousands to millions of fragments of DNA sequences are generated, aligned, sorted, and merged in order to construct a genomic sequence that approximates a chromosome in length. A step in this process may include comparing the DNA fragments to a reference sequence to determine where in the genome the fragments align.

[0018] A number of such steps are involved in building chromosome length sequences and in determining the variants of the sampled genetic sequence. Accordingly, a wide variety of methods have been developed for performing these steps. For instance, there exist commonly used software implementations for performing one or a series of such steps in a bioinformatics system. However, a common characteristic of such software based bioinformatics methods and systems is that they are labor intensive, take a long time to execute on general purpose processors, and are prone to errors.

[0019] A genomics and/or bioinformatics system, therefore, that could perform the algorithms implemented by such software in a less labor and/or processing intensive manner with a greater percentage accuracy would be useful. However, even as we approach the "$1000 Genome", the cost of analyzing, storing, and sharing this raw digital data has far outpaced the cost of producing it. This data analysis bottleneck is a key obstacle standing between these ever-growing raw data and the real medical insight we seek from it.

[0020] Accordingly, presented herein are systems, apparatuses, and methods for implementing a genomics and/or bioinformatic protocols, such as for performing one or more functions for analyzing genomic data, for instance, via software implementations and/or on an integrated circuit, such as on a hardware processing platform. For example, as set forth herein below, in various implementations, a combination of software implementable and/or hardware accelerator solutions, such as including an integrated circuit and software for interacting with the same, may be employed in performing such genomics and/or bioinformatics related tasks where the integrated circuit may be formed of one or more hardwired digital logic circuits, which may be interconnected by a plurality of physical electrical interconnects, that can be arranged as a set of processing engines, wherein each processing engine is capable of being configured to perform one or more steps in a bioinformatics genetic analysis protocol. An advantage of this arrangement is that the genomics and/or bioinformatics related tasks may be performed in a manner that is faster than the software alone such as typically engaged for performing such tasks. Such hardware accelerator technology, however, is currently not typically employed in the genomics and/or bioinformatics space.

SUMMARY

[0021] This present disclosure is related to performing a task such as in a genomics and/or bioinformatics protocol. In various instances, a plurality of tasks are performed, and in some instances these tasks are performed in a manner so as to form a pipeline, wherein each task and/or its substantial completion acts as a building block for each subsequent task until a desired end result is achieved. Accordingly, in various embodiments, the present disclosure is directed to performing one or more methods on one or more apparatuses wherein the methods and/or apparatuses has been optimized, such as for performing those methods. In certain embodiments, the one or more methods and/or one or more apparatuses are formulated into one or more systems.

[0022] For instance, in certain aspects, the present disclosure is directed to systems, apparatuses, and methods for implementing genomics and/or bioinformatic protocols such as, in various instances, for performing one or more functions for producing and/or analyzing genetic data employing innovative software and/or on an integrated circuit, such as implemented in a combination software and/or hardware processing platform. For example, in one aspect, a genomics and/or bioinformatics system is provided. The system may involve the performance of various genomics production and/or bioanalytical analysis functions that have been optimized so as to be performed faster and/or with increased accuracy. The methods for performing these functions may be implemented in software or hardware solutions.

[0023] Accordingly, in certain instances, methods are presented where the method involves the data production and/or acquisition and/or analysis that may include the performance of one or more algorithms where the algorithm(s) has been optimized in accordance with the manner, e.g., software, hardware, or a combination of both, in which it is to be implemented. In particular, where an algorithm is to be implemented in a software solution, the algorithm and/or its attendant processes, may be optimized so as to be performed faster and/or with better accuracy for execution by that media. Likewise, where the functions of an algorithm are to be implemented in a hardware solution, the hardware has been designed to perform these functions and/or their attendant processes in an optimized manner so as to be performed faster and/or with better accuracy for execution by that media. Further, where the functions involve a combination of both software and/or hardware solutions, these functions and their attendant processes have been designed and configured to work seamlessly together to achieve heretofore unattainable speed while maintaining or enhancing accuracy.

[0024] Hence, in one aspect, presented herein are systems, apparatuses, and methods for implementing genomics and/or bioinformatic protocols, such as for performing one or more functions for generating and/or analyzing genetic data, for instance, via one or more developed and/or optimized algorithms and/or on one or more optimized integrated circuits, such as on one or more hardware processing platforms. In one instance, methods are provided for implementing one or more algorithms for the performance of one or more steps for generating and/or analyzing genomic data, such as genetic sequence data, in a genomics and/or bioinformatics protocol. In another instance, methods are provided for implementing the functions of one or more algorithms for the performance of one or more steps for analyzing genomic data in a bioinformatics protocol, wherein the functions are at least partially implemented on an integrated circuit such as formed of one or more hardwired digital logic circuits. In such an instance, the hardwired digital logic circuits may be interconnected, such as by one or a plurality of physical electrical interconnects, and may be arranged in one or more subsets of hardwired digital logic circuits, which function as one or more processing engines. Accordingly, in various instances, a plurality of hardwired digital logic circuits are provided, which hardwired digital logic circuits are configured as a set of processing engines, wherein each processing engine is capable of performing one or more steps in a genomics and/or bioinformatics genetic production and/or analysis protocol, such as a bioinformatics processing pipeline.

[0025] More particularly, in one instance, a system for producing genetic sequence data, e.g., including devices and methods for nucleic acid sequencing, and/or for executing a sequence analysis pipeline on such genetic sequence data is provided. The system may include one or more of an electronic data source, such as associated with a DNA/RNA sequencing apparatus, such as herein described, a memory, and/or an integrated circuit. For instance, in one embodiment, an electronic data source is included, where in the electronic data source may be configured for generating and/or providing one or more digital signals, such as a digital signal representing one or more reads of genetic data, for example, where each read of genomic data includes a sequence of nucleotides. Further, the memory may be configured for storing one or more genetic reference sequences, one or more reference sequences, and may further be configured for storing an index, such as an index of the one or more genetic reference sequences and/or annotated splice junction data pertaining to the genetic sequences.

[0026] Further still, a device and/or method for producing genetic sequence data is provided. For example, an approach to DNA/RNA analysis, such as for genetic diagnostics and/or sequencing, involving one or more of nucleic acid hybridization, detection, and/or sequencing reactions is provided. In various instances, the approach may include hybridization and/or detection devices and/or procedures for implementing one or more of the following steps. Particularly, for genetic analysis, an RNA or DNA sample of a subject to be analyzed may be isolated and immobilized, e.g., directly and/or indirectly, on a substrate, such as a substrate containing a chemically sensitive one-dimensional (1-D) and/or two-dimensional (2D) reaction layer, and a probe of a known or to be detected genetic sequence, e.g., a disease marker, may be washed across the substrate, or vice versa. In various instances, one or more of the subject's DNA or RNA sample and/or the probe may be labeled.

[0027] In other instances, such as where the substrate includes a 1D or 2D, e.g., graphene, reaction layer, and/or other chemically sensitive reaction layer, a label or probe, such as a chemical or radioactive label may not be necessary and/or included. In either instance, if the disease marker is present, a binding event will occur, e.g., hybridization, and because the hybridization event is detectable, e.g., via a labeled analyte or probe and/or via the suitably configured reaction layer, as herein presented, the presence of the disease marker will be detected. If the disease marker is not present, there will be no reaction and therefore no detection. Of course, in some instances, the absence of a binding event may be the indicative event. Hence, the system may be configured such that the hybridization event may either be or not be detected thereby indicating the presence or absence of the disease marker in the subject's sample.

[0028] Likewise, for DNA and/or RNA sequencing, first, an unknown nucleic acid sequence the nucleotide identity of which is to be determined, e.g., a single-stranded sequence of DNA or RNA of a subject, is isolated, amplified, and immobilized on a substrate, which, as described herein may include a 1D, 2D, e.g., graphene layer, and/or other configured reaction layer thereon. Next, a known nucleic acid, e.g., a nucleotide base, which may be labeled with an identifiable tag is contacted with the unknown nucleic acid sequence in the presence of a polymerase. As noted, where the reaction event occurs proximate a suitably configured reaction layer, e.g., a graphene containing reaction layer, a tagged or labeled reactant need not be included. Hence, when hybridization occurs, the nucleotide binds to its complementary base in the unknown sequence, e.g., the sample DNA or RNA being sequenced, and is thus immobilized to the nucleic acid sample sequence on the surface of the substrate, such as proximate the reaction layer. The binding event can then be detected, e.g., optically, electrically, and/or via a suitably detectable reaction occurring at the reaction layer. These steps are then repeated until the entire DNA or RNA sample has been completely sequenced. Typically, these steps are performed by a Next Gen Sequencer, as is known in the art, or they may be performed in accordance with the devices and methods herein described, such that thousands to millions of sequencing reactions may be performed and/or processed concurrently and digital data produced as a result thereof may be streamed and/or analyzed in conjunction with the innovative sequencing devices and processes disclosed herein such as in a multiplex genomics and/or bioinformatics processing pipeline.

[0029] For instance, in one aspect, such as with respect to the innovative sequencing devices herein presented, an appropriately configured sequencing platform may be provided as a field effect transistor (FET) containing a chemical reaction layer such as for use in performing a hybridization and/or sequencing reaction. Particularly, such a field effect transistor (FET) may be fabricated on a primary structure, such as a wafer, e.g., a silicon wafer. In various instances, the primary structure may include one or more additional structures, for instance, in a stacked configuration, such as an insulator material layer. For example, an insulator material may be included on top of the silicon wafer primary structure, and may be an inorganic material, such as a silicon oxide, e.g., a silicon dioxide, or a silicon nitride, or an organic material, such as a polyimide, BCB, or other like material.

[0030] The primary structure and/or insulator layer may include a further structure containing one or more of a conductive source and/or a conductive drain, such as separated one from another by a space, e.g., forming a channel, and embedded in the primary structure and/or insulator material layer and/or may be planar with a top and/or bottom surface of the insulator so as to form a top and/or bottom gate. In various instances, the structures, e.g., the silicon wafer and accompanying structure(s), may further include or otherwise be associated with an integrated circuit, such as a processor, e.g., a microprocessor, for processing the generated data, such as sensor derived data, e.g., data derived as a result of a sequencing reaction, e.g., proximate the gate region. Accordingly, the plurality of structures may be configured as, or otherwise include, an integrated circuit, and/or may be present as an ASIC, a structured ASIC, or an FPGA.

[0031] Particularly, one or more of these structures may be configured as a complementary metal-oxide semiconductor (CMOS), which in turn may be configured as a chemically-sensitive FET sensor containing one or more of a conductive source, a conductive drain, and/or a reaction region, such as a gate region, which itself may include a micro- or nano-channel, chamber, and/or well configuration, which sensor may be adapted so as to communicate with a processor. For instance, the FET may include a CMOS configuration having or otherwise being associated with an integrated circuit that is fabricated on a silicon wafer, which further includes an insulator layer, which insulator layer includes a conductive source and a conductive drain embedded in the insulator layer, which source and drain may be composed of metal, such as a damascene copper. In various instances, the CMOS and relevant structures may include a surface, e.g., a top surface, which surface may include a channel and/or a chamber so as to form a reaction well where the surface of the reaction well may be configured to extend, e.g., lengthwise, from the conductive source to the conductive drain and be adapted to receive various reagents instrumental in performing a biochemical reaction, such as a DNA or RNA hybridization and/or sequencing reaction.

[0032] In certain instances, the surface and/or channel and/or chamber may include a one-dimensional transistor material, a two-dimensional transistor material, a three-dimensional transistor material, and/or the like. In various instances, a one-dimensional (1D) transistor material may be included, which 1D material may be composed of one or more, e.g., a plurality of carbon nanotube(s) or semiconductor nanowire(s), which in various instances may be formed as a sheet or a channel, and/or in various instances may include a nanopore, although in many instances, a nanopore is not included nor necessary. In various instances, a two-dimensional (2D) transistor material may be included, which 2D material may include a graphene layer, silicene, molybdenum disulfide, black phosphorous, metal dichalcogenides, and/or other such materials. A three-dimensional (3D) configuration may also be present. In various instances, the surface and/or channel may include a dielectric layer. Additionally, in various instances, a reaction layer, e.g., an oxide layer, may be disposed on the surface and/or within the channel and/or chamber, such as layered or otherwise deposited on the 1D, 2D, e.g., graphene, or 3D layer(s). Such an oxide layer may be an aluminum oxide or a silicon oxide, such as silicon dioxide. In various instances, a passivation layer may be disposed on the surface and/or channel and/or within the chamber, such as layered or otherwise deposited on the 1D, 2D, e.g., graphene, or 3D layer(s) and/or on an associated reaction layer on the surface and/or channel and/or chamber.

[0033] Accordingly, in particular instances, the primary and/or secondary and/or tertiary structures of the transistor may be fabricated or otherwise configured so as to include a chamber or well structure in and/or on the surface of the substrate, e.g., in a manner so as to form a reaction region. For instance, a well structure may be positioned on a portion of a surface, e.g., an exterior surface, of the primary and/or secondary and/or tertiary structures. In some instances, the well structure may be configured as a micro- or nano-chamber and may be formed on top of, or may otherwise include, at least a portion of the 1D, 2D, e.g., graphene, and/or 3D material, and/or may additionally include the reaction, e.g., oxide, and/or passivation layers. In various instances, the chamber and/or well structure may define an opening, such as an opening that allows access to an interior of the chamber, such as allowing direct contact with the 1D, e.g., carbon nanotube or nanowire, 2D, e.g., graphene, or 3D surface and/or channel and/or chamber. In particular instances, the chamber and/or well may be dimensioned so as to be a micro- or nano-chamber.

[0034] Accordingly, a further aspect of the present disclosure is a bio-sensor such as for performing a nucleic acid sequencing reaction. The bio-sensor includes a CMOS structure that may be configured as a chemically sensitive FET sensor and may include a metal containing source and drain, e.g., a damascene copper source and/or drain, that further includes a surface, such as a reaction region that includes a 1D or 2D layered, e.g., a graphene layered, or 3D surface that extends from the source to the drain. Particularly, the reaction region may include or otherwise be configured as a well or chamber structure that may be positioned on or in a portion of an exterior surface of the 1D, 2D, or 3D layered well. In such an instance, the well structure may be configured so as to define an opening that allows for direct contact with the nanotube, nanowire, and/or graphene well or chamber surface. In various instances, an oxide and/or passivation layer may be disposed in or on the chamber surfaces. Hence, in certain instances, a chemically-sensitive transistor, such as a field effect transistor (FET) including one or more nano- or micro-wells for performing a sequencing reaction may be provided.

[0035] In some embodiments, the chemically-sensitive field effect transistor may include a plurality of wells and may be configured as an array, e.g., a sensor array. Such an array or arrays may be employed such as to detect a presence and/or concentration change of various analyte types in a wide variety of chemical and/or biological processes, including DNA and/or RNA hybridization and/or DNA and/or RNA sequencing reactions. For instance, the devices herein described, and/or systems including the same, may be employed in a method for the analysis of biological or chemical materials, such as for whole genome sequencing and/or analysis, genome typing analysis, micro-array analysis, panels analysis, exome analysis, micro-biome analysis, and/or clinical analysis, such as cancer analysis, NIPT analysis, and/or UCS analysis and the like.

[0036] Hence, in a particular embodiment, a graphene FET (gFET) array may be employed to facilitate DNA and/or RNA sequencing and processing techniques, such as in a genetic analysis pipeline-on-a-chip, as herein described. For example, a CMOS FET, e.g., a graphene FET (gFET) array, may be configured to include a reaction well that includes a reaction layer that is adapted to detect changes in hydrogen ion concentration (pH), changes in other analyte concentrations, and/or binding events associated with chemical processes such as related to DNA and/or RNA synthesis, such as within a gated reaction chamber or well of the gFET based sensor. Such a chemically-sensitive field effect transistor may include or be adapted to associate with one or more integrated circuits and/or be adapted to increase the measurement sensitivity and/or accuracy of the sensor and/or associated array(s), such as by including one or more surfaces within the reaction chamber or well having at least one surface layered with a 1D and/or 2D and/or 3D material, a dielectric or reaction layer, a passivation layer, and/or the like.

[0037] Accordingly, an aspect of the present disclosure may include one or more integrated circuits that may be formed of one or more sets of hardwired digital logic circuits, such as where a set of the hardwired digital logic circuits are interconnected, e.g., by a plurality of physical electrical interconnects or interfaces, and may be adapted so as to participate in the performance and/or detection of a DNA or RNA hybridization and/or sequencing reaction, e.g., primary processing, and/or may further be adapted for processing the results thereof, e.g., such as in one or more secondary and/or tertiary processing steps. In such instances, the integrated circuit may include an input, such as via one or more of the plurality of physical electrical interconnects or interfaces, so as to be connected with an electronic data generating source, such as a sequencing CMOS FET of the disclosure and/or a Next Gen Sequencer, which is configured for generating such data, e.g., in the form of a plurality of sequenced segments, e.g., reads, of genomic data. In particular instances, the one or more integrated circuits may include a set of hardwired digital logic circuits that are configured for performing a secondary and/or tertiary processing analysis pipeline on the generated reads of genomic data, and may therefore be connected to the electronic data generating source such as through the one or more of the associated interconnects or interfaces.

[0038] In such an instance, the hardwired digital logic circuits of the integrated circuit and/or associated interconnects and/or interfaces may be configured so as to be able to receive the one or more reads of genomic data, e.g., from the electronic data source. In particular instances, one or more of the hardwired digital logic circuits may be arranged as a set of processing engines, such as where each processing engine is formed of a subset of the hardwired digital logic circuits, and is configured so as to perform one or more steps in the sequencing and/or analysis pipeline, such as on the plurality of reads of genomic data. In such instances, each subset of the hardwired digital logic circuits may, in certain instances, be in a wired, e.g., hardwired, configuration so as to perform the one or more steps in the sequence and/or analysis pipeline. However, as indicated above, one or more of the steps in the sequence and/or analysis pipeline may be configured so as to be implemented in software, such as where the software and/or hardware have been adapted to operate in an optimized manner with respect to each other.

[0039] Accordingly, in various instances, a plurality of hardwired digital logic circuits are provided wherein the hardwired digital logic circuits are arranged as a set of processing engines, wherein one or more of the processing engines may include one or more of a sequencing module and/or a mapping module and/or an alignment module and/or a sorting module and/or variant call module and/or one or more tertiary processing modules as herein described. For instance, in various embodiments, the one or more of the processing engines may include a mapping module, which mapping module may be in a wired configuration and further be configured for communicating with a memory or other suitably configured register, on the device or otherwise associated therewith, e.g., via a suitably configured interconnect or interface, so as to access an index containing one or more of a genetic reference sequence(s), one or more reads of generated sequencing data, and/or a splice junction index (e.g., in the case of RNA sequencing), and employing the same so as to perform one or more mapping operations.

[0040] Particularly, a suitably configured processing engine(s) may include or may otherwise be adapted as a mapping module for performing one or more mapping operations, such as including accessing an index of the one or more genetic reference sequences from the memory, such as by one or more of the plurality of physical electronic interconnects, for example, so as to map the plurality of reads to one or more segments of the one or more genetic reference sequences. Additionally, in various embodiments, the one or more of the processing engines may include an alignment module, which alignment module may be in the wired configuration and may be configured for accessing the one or more genetic reference sequences from the memory, such as by one or more of the plurality of physical electronic interconnects, for example, so as to align the plurality of reads to the one or more segments of the one or more genetic reference sequences.

[0041] Further, in various embodiments, the one or more of the processing engines may include a sorting module, which sorting module may be in the wired configuration and may be configured for accessing the one or more aligned reads from the memory, such as by one or more of the plurality of physical electronic interconnects, for example, so as to sort each aligned read, such as according to its one or more positions in the one or more genetic reference sequences. In such instances, the one or more of the plurality of physical electrical interconnects may include an output from the integrated circuit, such as for communicating result data from the mapping module and/or the alignment module and/or the sorting module. Furthermore, in particular embodiments, as indicated above, one or more of the processing engines may be configured for interacting with various software and/or hardware implemented processing functions, such as via one or more interconnects, e.g., a plurality of physical electronic interconnects, for performing one or more steps in the analysis pipeline including implementing one or more of DNA and/or RNA sequencing protocols and/or a variant call protocol.

[0042] In various instances, the one or more integrated circuit(s) may include a master controller so as to establish the wired configuration for each subset of the hardwired digital logic circuits, for instance, for performing the one or more of mapping, aligning, sorting, and/or variant calling function, which functions may be configured as one or more steps in a sequence analysis pipeline and/or may include the performance of one or more aspects of a sequencing and/or variant call function. Further, in various embodiments, the one or more integrated circuits herein disclosed may be configured as a field programmable gate array (FPGA) having hardwired digital logic circuits, such as where the wired configuration may be established upon manufacture of the integrated circuit, and thus may be non-volatile. In other various embodiments, the integrated circuit may be configured as an application specific integrated circuit (ASIC) having hardwired digital logic circuits. In other various embodiments, the integrated circuit may be configured as a structured application specific integrated circuit (Structured ASIC) having hardwired digital logic circuits.

[0043] In certain instances, the CMOS FET sequencing and/or biosensor as well as the one or more integrated circuits and/or associated memories may be housed on an expansion card, such as a peripheral component interconnect (PCI) card, for instance, in various embodiments, an integrated circuit(s) of the disclosure may be a chip that is associated with a PCIe card. In various instances, the integrated circuit and/or chip may be a component within a sequencer, such as an automated sequencer employing a FET sensor array and/or an NGS, and/or in other embodiments, the integrated circuit and/or expansion card may be accessible via the internet, e.g., via the cloud. Further, in some instances, the memory may be a volatile random access memory (RAM) or DRAM.

[0044] Accordingly, in one aspect, an apparatus for executing one or more steps of a sequence analysis pipeline, such as on genetic data, is provided wherein the genetic data includes one or more of a genetic reference sequence(s), an index of the one or more genetic reference sequence(s), an index of one or more splice junctions, e.g., an annotated splice junction index or table, and/or a plurality of reads, such as of genetic data, e.g., DNA or RNA. In various instances, the apparatus may include an integrated circuit, which integrated circuit may include one or more, e.g., a set, of hardwired digital logic circuits, wherein the set of hardwired digital logic circuits may be interconnected, such as by one or a plurality of physical electrical interconnects or other suitably configured interfaces. In certain instances, the one or more of the plurality of physical electrical interconnects may include an input, such as for receiving the plurality of reads of genomic data, such as from a sequencing device as disclosed herein. Additionally, the set of hardwired digital logic circuits may further be in a wired configuration, so as to access the index of the one or more genetic reference sequences and/or annotative splice junctions, via one of the plurality of physical electrical interconnects, and to map the plurality of reads of DNA and/or RNA to one or more segments of the one or more genetic reference sequences, such as according to the index or indexes.

[0045] In various embodiments, the index may include one or more hash tables, such as a primary and/or secondary hash table and/or a splice junction table. For instance, a primary hash table may be included, wherein in such an instance, the set of hardwired digital logic circuits may be configured to do one or more of: extracting one or more seeds of genetic data from the plurality of reads of genetic data; executing a primary hash function, such as on the one or more seeds of genetic data so as to generate a lookup address for each of the one or more seeds; and accessing the primary hash table using the lookup address so as to provide a location in the one or more genetic reference sequences for each of the one or more seeds of genetic data. In various instances, the one or more seeds of genetic data may have a fixed number of nucleotides.

[0046] Further, in various embodiments, the index may include a secondary hash table, such as where the set of hardwired digital logic circuits is configured for at least one of extending at least one of the one or more seeds with additional neighboring nucleotides, so as to produce at least one extended seed of genetic data; executing a hash function, e.g., a secondary hash function, on the at least one extended seed of genetic data, so as to generate a second lookup address for the at least one extended seed; and accessing the secondary hash table, e.g., using the second lookup address, so as to provide a location in the one or more genetic reference sequences for each of the at least one extended seed of genetic data. In various instances, the secondary hash function may be executed by the set of hardwired digital logic circuits, such as when the primary hash table returns an extend record instructing the set of hardwired digital logic circuits to extend the at least one of the one or more seeds with the additional neighboring nucleotides. In certain instances, the extend record may specify the number of additional neighboring nucleotides by which the at least one or more seeds is extended, and/or the manner in which the seed is to be extended, e.g., equally by an even number of "x" nucleotides to each end of the seed. And thus, the various generated DNA reads may be mapped against a referent sequence.

[0047] Furthermore, as is known, DNA codes for genes. However, in order for a gene to be expressed, its genetic code needs to be transcribed and translated into proteins. Specifically, a gene may be transcribed within the nucleus of a cell by RNA polymerase enzymes into a messenger RNA (mRNA) transcript or other types of RNA (e.g., a transfer RNA). The immediate RNA transcript is a single-stranded copy of the gene, except that DNA thymine (T) bases are transcribed into RNA Uracil (U) bases. But immediately after this copy is produced, its sequence includes both various intron- and exon copies, where the various intron-copies usually need to be spliced out, e.g., by spliceosomes, leaving only the exon-copies that are to be concatenated together at "splice junctions" (which are not thereafter directly evident), so as to form codon regions. Spliced mRNA containing the codon regions is then transported out of the cellular nucleus to a ribosome, which decodes it into a protein, where each group of three RNA nucleotides form the codon that codes for one amino acid. During the decoding process, a string of amino acids are strung together, and when strung together and glycosylated form the proteins, of which the cells, tissues, and organs of the body are composed. In this manner, genes in DNA serve as original instructions for the manufacture of proteins.

[0048] Accordingly, because the DNA includes both coding regions, i.e., exons, and non-coding regions, i.e. introns, the mapping and/or aligning and/or sorting of RNA back to its genetic precursor in the genomic DNA, may be complicated. Particularly, each gene exists on a single strand of the double-stranded DNA double-helix, often as a series of exons (coding segments) separated by introns (non-coding segments). Some genes have only a single exon, but most have several exons (separated by introns), and some have hundreds of exons or thousands of exons. Exons are commonly a few hundred nucleotides long, but may be as short as a single nucleotide or as long as tens or hundreds of thousands. Introns are commonly thousands of nucleotides long, and some exceed a million nucleotides. Hence, when mapping, aligning, and/or sorting from RNA, e.g., spliced mRNA, portions of the spliced mRNA may come from different regions of the DNA that may be separated from each other by one or two or even a million or more nucleotides. This makes the processing of RNA very complicated.

[0049] However, an aspect of the present disclosure overcomes these challenges, by the methods herein described, and therefore allows for the rapid and accurate whole-transcriptome RNA sequencing, mapping, aligning, and/or sorting. More particularly, where RNA processing is involved, the aforementioned index may include one or more tables, e.g., a hash table or other index, which may be formed, e.g., as set forth above, and may include or otherwise be associated with a table that allows for the ready lookup of various known and/or determined splice junctions employed by biological systems in transcribing RNA from DNA, as described in detail herein below. In such instances, therefore, an RNA-capable mapper/aligner may be configured to process such splice junctions and account for RNA-sequence reads that correspond to segments of transcribed and spliced RNA, such as where the read crosses one or more splice junctions; which, with respect to the DNA-oriented reference genome, means a first portion of the read came from, and should map to, a first exon, and a second portion of the read should map to a second exon, and so forth. Accordingly, the index may include or otherwise be associated with one or more splice junction tables and the set of hardwired digital logic circuits may be configured to do one or more of: employing said splice junction data to determine and/or extract one or more seeds of genetic, e.g., RNA, data from the plurality of reads of genetic RNA data; executing a function, e.g., a hash function, such as on the one or more seeds of genetic RNA data, as described herein, so as to generate a lookup address for each of the one or more seeds; and accessing the hash table using the lookup address so as to provide a location in the one or more genetic reference sequences for each of the one or more seeds of genetic RNA data.

[0050] Additionally, in one aspect, an apparatus for executing one or more steps of a sequence analysis pipeline on genetic sequence data, e.g., either DNA or RNA, is provided, wherein the genetic sequence data includes one or more genetic reference sequences, which may include both exons and introns, an index of the one or more genetic reference sequences, and/or an index of annotated splice junctions, and a plurality of reads of genomic data. In various instances, the apparatus may include an integrated circuit, which integrated circuit may include one or more, e.g., a set, of hardwired digital logic circuits, wherein the set of hardwired digital logic circuits may be interconnected, such as by one or a plurality of physical electrical interconnects. In certain instances, the one or more of the plurality of physical electrical interconnects may include an input, such as for receiving the plurality of reads of genomic data, which reads may have previously been processed, as herein described so as to be mapped. Additionally, the set of hardwired digital logic circuits may further be in a wired configuration, so as to access the one or more mapped genetic reference sequences, via one of the plurality of physical electrical interconnects, to receive location information, e.g. such as from a mapper, specifying one or more segments of the one or more reference sequences, and to align the plurality of reads to the one or more segments of the one or more genetic reference sequences.

[0051] Accordingly, in various instances, the wired configuration of the set of hardwired digital logic circuits, are configured to align the plurality of reads of DNA or RNA genetic data to the one or more segments of the one or more genetic reference sequences. In the performance of such an alignment function, a set of the hardwired digital logic circuits may be configured to include or otherwise function as a wave front processor that me be formed of the wired configuration of the set of hardwired digital logic circuits. In certain embodiments, the wave front processor may be configured to process an array of virtual cells of an alignment matrix, such as a virtual matrix defined by a subset of the set of hardwired digital logic circuits. For instance, in certain instances, the alignment matrix may define a first axis, e.g., representing one of the plurality of reads, and a second axis, e.g., representing one or more of the segments of the one or more genetic reference sequences. In such an instance, the wave front processor may be configured to generate a wave front pattern of cells that extend across the array of cells from the first axis to the second axis; and may further be configured to generate a score, such as for each cell in the wave front pattern of cells, which score may represent the degree of matching, on a base by base basis, of the one of the plurality of reads and the one of the segments of the one or more genetic reference sequences.

[0052] In such an instance, the wave front processor may further be configured so as to steer the wave front processing pattern of cells over the alignment matrix such that the highest score may be centered on the wave front pattern of cells. Additionally, in various embodiments, the wave front processor may further be configured to backtrace one or more, e.g., all, the positions in the scored wave front pattern of cells through previous positions in the alignment matrix; track one or more, e.g., all, of the backtraced paths until a convergence is generated; and generate a CIGAR string based on the backtrace from the convergence.

[0053] In certain embodiments, the wired configuration of the set of hardwired digital logic circuits to align the plurality of reads to the one or more segments of the one or more genetic reference sequences may include a wired configuration to implement a Burrows-Wheeler algorithm, as described herein, e.g., for mapping prior to aligning, and/or to implement a Smith-Waterman and/or Needleman-Wunsch scoring algorithm, such as for aligning. In such an instance, the Smith-Waterman and/or Needleman-Wunsch scoring algorithm may be configured to implement a scoring parameter that is sensitive to base quality scores. Further, in certain embodiments, the Smith-Waterman scoring algorithm may be an affine Smith-Waterman scoring algorithm.

[0054] In particular embodiments, the apparatus may include an integrated circuit, which integrated circuit may include one or more, e.g., a set, of hardwired digital logic circuits, wherein the set of hardwired digital logic circuits may be interconnected, such as by one or a plurality of physical electrical interconnects. In certain of these instances, the one or more of the plurality of physical electrical interconnects may include an input, such as for receiving the plurality of reads of genomic data, which reads may have previously been processed, as herein described so as to be mapped and/or aligned. Additionally, the set of hardwired digital logic circuits may further be in a wired configuration, so as to access the one or more genetic reference sequences, via one of the plurality of physical electrical interconnects, to receive location information, e.g. such as from a mapper and/or aligner, specifying one or more segments of the one or more reference sequences, and to sort the plurality of reads to the one or more segments of the one or more genetic reference sequences.

[0055] In view of the above, in one aspect, a method for sequencing genetic material, e.g., so as to produce electronic genetic data, may be provided. In particular instances, the method involves the use of a Next Gen Sequencer for sequencing of genomic DNA and/or RNA derived therefrom, as described generally herein and known in the art. In other instances, the method involves the use of a Next Gen Sequencer, modified as described herein, for sequencing of genomic DNA and/or RNA derived therefrom. In further instances, the method involves the use of a Field Effect Transistor and/or CMOS Sequencer, e.g., a sequencer-on-a-chip, as described herein in detail below, for the sequencing of genomic DNA and/or RNA derived therefrom. In various instance, the genetic material once produced may be converted into an electronic form, e.g., a digital form, that may be streamed or otherwise transferred to one or more of the pipeline modules herein described.

[0056] Accordingly, once the electronic, e.g., analog or digital, genetic data, such as sequencing data, is produced, transferred, and received, another aspect of the disclosure is directed to executing a sequence analysis pipeline on such genetic sequence data. The genetic data may include one or more genetic reference sequences, one or more indexes of the one or more genetic reference sequences and/or a list of one or more annotated splice junctions (e.g., in the case of RNA sequencing) pertaining thereto, and/or a plurality of reads of genomic data (e.g., DNA and/or RNA sequence segments). The method may include one or more of receiving, accessing, mapping, aligning, sorting, and/or performing a variant call function on various iterations of the genetic sequence data. For instance, in certain embodiments, the method may include receiving, on an input to an integrated circuit from an electronic data source, one or more of a plurality of reads of genomic data, wherein each read of genomic data may include a sequence of nucleotides. In such an instance, the integrated circuit may be formed of a set of hardwired digital logic circuits that are formed or otherwise configured as one or more processing engines such as are interconnected by a plurality of physical electrical interconnects, which physical electrical interconnects may include one or more of the plurality of physical electrical interconnects comprising the input.

[0057] The method may further include accessing, by the integrated circuit on one or more of the plurality of physical electrical interconnects from a memory, the index of the one or more genetic reference sequences and/or, in the case of RNA sequencing, the annotated splice junctions. Particularly, if annotated splice junctions are provided to the mapper engine, they can be leveraged to improve mapping sensitivity. In such an instance, the list of annotated junctions may be loaded into the memory so as to be accessible by the mapper engine so as to assist with the mapping of RNA genetic material. Advantageously, the annotated junctions may be formatted into a table, e.g., a hash table or index that may be associated therewith, so as to be easily accessed by the mapper engine. Accordingly, the method may include mapping, by a first subset of the hardwired digital logic circuits of the integrated circuit, the plurality of genetic reads, e.g., DNA or RNA reads, to one or more segments of the one or more genetic reference sequences. Additionally, the method may include accessing, by the integrated circuit on one or more of the plurality of physical electrical interconnects from the memory, the one or more mapped reads and/or genetic reference sequences; and aligning, e.g., by a second subset of the hardwired digital logic circuits of the integrated circuit, the plurality of reads, e.g., mapped reads, to the one or more segments of the one or more genetic reference sequences.

[0058] In various embodiments, the method may additionally include accessing, by the integrated circuit on one or more of the plurality of physical electrical interconnects from a memory, the aligned plurality of reads. In such an instance the method may include sorting, by a third subset of the hardwired digital logic circuits of the integrated circuit, the aligned plurality of reads according to their positions in the one or more genetic reference sequences. In certain instances, the method may further include outputting, such as on one or more of the plurality of physical electrical interconnects of the integrated circuit, result data from the mapping and/or the aligning and/or the sorting, such as where the result data includes positions of the mapped and/or aligned and/or sorted plurality of reads.

[0059] Further, once the genetic data has been generated and/or processed, e.g., in one or more secondary processing protocols, such as by being mapped, aligned, and/or sorted, the resultant mapped, aligned, and/or sorted date may then be subjected to a variant call procedure, as herein described, so as to produce one or more variant call files, for instance, to determine how the genetic sequence data from a subject differs from one or more reference sequences. Additionally, a further aspect of the disclosure may be directed to performing one or more other analytical functions on the generated and/or processed genetic data such as for further, e.g., tertiary, processing. For example, the system may be configured for further processing of the generated and/or secondarily processed data, such as by running it through one or more tertiary processing pipelines, such as one or more of a genome pipeline, an epigenome pipeline, metagenome pipeline, joint genotyping, a MuTect2 pipeline, or other tertiary processing pipeline, such as by the devices and methods disclosed herein. Particularly, in various instances, an additional layer of processing may be provided, such as for disease diagnostics, therapeutic treatment, and/or prophylactic prevention, such as including NIPT, NICU, Cancer, LDT, AgBio, and other such disease diagnostics, prophylaxis, and/or treatments employing the data generated by one or more of the present primary and/or secondary and/or tertiary pipelines. Hence, the devices and methods herein disclosed may be used to generate genetic sequence data, which data may then be used to generate one or more variant call files and/or other associated data that may further be subject to the execution of other tertiary processing pipelines in accordance with the devices and methods disclosed herein, such as for particular and/or general disease diagnostics as well as for prophylactic and/or therapeutic treatment and/or developmental modalities.

[0060] Hence, in various instances, implementations of various aspects of the disclosure may include, but are not limited to: apparatuses, systems, and methods including one or more features as described in detail herein, as well as articles that comprise a tangibly embodied machine-readable medium operable to cause one or more machines (e.g., computers, etc.) to result in operations described herein. Similarly, computer systems and/or networks are also described that may include one or more processors and/or one or more memories coupled to the one or more processors, either directly or remotely. Accordingly, computer implemented methods consistent with one or more implementations of the current subject matter can be implemented by one or more data processors residing in a single computing system or multiple computing systems, such as one or more computer clusters. Such multiple computing systems can be connected and can exchange data and/or commands or other instructions or the like via one or more connections, including but not limited to a connection over a network (e.g. the Internet, a wireless wide area network, a local area network, a wide area network, a wired network, or the like), via a direct connection between one or more of the multiple computing systems, etc. A memory, which can include a computer-readable storage medium, may include, encode, store, or the like one or more programs that cause one or more processors to perform one or more of the operations described herein.

[0061] The details of one or more variations of the subject matter described herein are set forth in the accompanying drawings and the description below. Other features and advantages of the subject matter described herein will be apparent from the description and drawings, and from the claims. While certain features of the currently disclosed subject matter are described for illustrative purposes in relation to an enterprise resource software system or other business software solution or architecture, it should be readily understood that such features are not intended to be limiting. The claims that follow this disclosure are intended to define the scope of the protected subject matter.

DESCRIPTION OF DRAWINGS

[0062] The accompanying drawings, which are incorporated in and constitute a part of this specification, show certain aspects of the subject matter disclosed herein and, together with the description, help explain some of the principles associated with the disclosed implementations. In the drawings,

[0063] FIG. **1** depicts an RNA read, illustrating the crossover between one or more splice junctions, and a seed crossing the read's splice junction

[0064] FIG. **2** depicts another exemplary RNA read, illustrating that short (L-base) seeds can be configured to more easily fit into short exons, and accommodate short exon overhangs, or exon segments cut by edits such as SNPs.

[0065] FIG. **3** depicts an exemplary reference bins that are within the search range of successfully-mapped K-base seeds that can be queried in the anchored-seed hash table, such as using L-base seeds.

[0066] FIG. **4** depicts a comparison of read portions left and right of a stitch position.

[0067] FIG. **5** depicts an abstract alignment rectangle, with concatenated query sequence on the vertical axis and concatenated reference sequence on the horizontal axis.

[0068] FIG. **6** depicts an HMM 3-state based model illustrating the transition probabilities of going from one state to another.

[0069] FIG. **7** depicts an exemplary HMM matrix showing an anti-diagonal processing wavefront or swath.

[0070] FIG. **8** an exemplary cell to be processed in the HMM matrix of FIG. **7** and showing the data dependencies employed in calculating the transition state of the demarcated cell.

[0071] FIG. **9** depicts another exemplary matrix, this time with a horizontal processing swath.

[0072] FIG. **10** depicts the exemplary cell of FIG. **8** showing the cycle dependencies with respect to the processing of the demarcated cell.

[0073] FIG. **11** depicts an exemplary output end for a cell at the end of a pipeline in the matrix of FIG. **7**.

[0074] FIG. **12** depicts a histogram of an HMM table.

[0075] FIG. **13** depicts a high-level view of an integrated circuit of the disclosure including a HMM interface structure.

[0076] FIG. **14** depicts the integrated circuit of FIG. **13**, showing an HMM cluster features in greater detail.

[0077] FIG. **15** depicts an overview of HMM related data flow throughout the system including both software and hardware interactions.

[0078] FIG. **16** depicts exemplary HMM cluster collar connections.

[0079] FIG. **17** depicts an exemplary HMM engine HMEM organization.

[0080] FIG. **18** depicts an exemplary HMM engine RMEM organization.

[0081] FIG. **19** depicts a high-level view of the major functional blocks within an exemplary HMM hardware accelerator.

[0082] FIG. **20** depicts an exemplary HMM matrix structure and hardware processing flow.

[0083] FIG. **21** depicts an enlarged view of a portion of FIG. **20** showing the data flow and dependencies between nearby cells in the HMM M, I, and D state computations within the matrix.

[0084] FIG. **22** depicts exemplary computations useful for M, I, D state updates.

[0085] FIG. **23** depicts M, I, and D state update circuits, including the effects of simplifying assumptions of FIG. **22** related to transition probabilities and the effect of sharing some M, I, D adder resources with the final sum operations.

[0086] FIG. **24** depicts Log domain M, I, D state calculation details.

[0087] FIG. **25** depicts an HMM state transition diagram showing the relation between GOP, GCP and transition probabilities.

[0088] FIG. **26** depicts an HMM Transprobs and Priors generation circuit to support the general state transition diagram of FIG. **25**.

[0089] FIG. **27** depicts a simplified HMM state transition diagram showing the relation between GOP, GCP and transition probabilities.

[0090] FIG. **28** depicts a HMM Transprobs and Priors generation circuit to support the simplified state transition diagram of FIG. **25**.

[0091] FIG. **29** depicts an exemplary theoretical HMM matrix and illustrates how such an HMM matrix may be traversed.

[0092] FIG. **30** depicts a block diagram for a genomic infrastructure for onsite and/or cloud based genomics processing and analysis.

[0093] FIG. **31**A depicts a block diagram of a local and/or cloud based computing function of FIG. **30** for a genomic infrastructure for onsite and/or cloud based genomics processing and analysis.

[0094] FIG. **31**B depicts the block diagram of FIG. **31**A illustrating greater detail regarding the computing function for a genomic infrastructure for onsite and/or cloud based genomics processing and analysis.

[0095] FIG. **31**C depicts the block diagram of FIG. **31** illustrating greater detail regarding the 3^{rd}-Party analytics function for a genomic infrastructure for onsite and/or cloud based genomics processing and analysis.

[0096] FIG. **32**A depicts a block diagram illustrating a hybrid cloud configuration.

[0097] FIG. **32**B depicts the block diagram of FIG. **32**A in greater detail, illustrating a hybrid cloud configuration.

[0098] FIG. **32**C depicts the block diagram of FIG. **32**A in greater detail, illustrating a hybrid cloud configuration.

[0099] FIG. **33** depicts a block diagram illustrating a primary, secondary, and/or tertiary analysis pipeline as presented herein.

[0100] FIG. **34** depicts a flow diagram for an analysis pipeline of the disclosure.

[0101] FIG. **35** illustrates an exemplary design and fabrication of an integrated circuit.

[0102] FIG. **36** is a block diagram of a hardware processor architecture in accordance with an implementation of the disclosure.

[0103] FIG. **37** is a block diagram of a hardware processor architecture in accordance with another implementation.

[0104] FIG. **38** is a block diagram of a hardware processor architecture in accordance with yet another implementation.

[0105] FIG. **39** illustrates a genetic sequence analysis pipeline.

[0106] FIG. **40** illustrates processing steps using a genetic sequence analysis hardware platform.

[0107] FIG. **41** illustrates an apparatus in accordance with an implementation of the disclosure.

[0108] FIG. **42** illustrates another apparatus in accordance with an alternative implementation of the disclosure.

[0109] FIG. **43** illustrates a genomics processing system in accordance with an implementation.

[0110] When practical, similar reference numbers denote similar structures, features, or elements.

DETAILED DESCRIPTION

[0111] To address these and potentially other issues with the currently available solutions, methods, systems, articles of manufacture, and the like, consistent with one or more implementations of the current subject matter can, among other possible advantages, provide a sequencing and/or a sequence analysis apparatus for executing a sequence analysis pipeline on genetic sequence data. The following provides details of various implementations of a sequencing platform, a sequence analysis pipeline, as well as a system for performing one or more tertiary processing protocols.

[0112] In its most basic form, the body is comprised of cells containing proteins, various of the proteins between cells connect the cells to form tissues, tissues form organs, organs form systems, and these systems function together to ensure the body operates to sustain the life of the individual. The cells of the body, therefore, are the building blocks of life. More particularly, each cell has a nucleus, and within the nucleus of every cell reside chromosomes. Chromosomes are formed from Deoxyribonucleic Acid (DNA), which has an organized but winding double helix structure. The DNA itself is comprised of two opposed, but complementary strands of nucleotides, which nucleotides comprise the genes that code for the proteins that give the cells their structures and mediate the functions and regulations of the body's tissues and organs. Basically, proteins do most of the work of cells in maintaining the body's normal processes and functions.

[0113] Given the multiplicity of components of the body and the complexity involved in how they interact with one another to maintain the body's various processes and functions, there are a multiplicity of ways that the body may malfunction on any one of these different levels. For instance, in one such instance, there may be a malfunction in the way a particular gene codes for a given protein, which dependent on the protein and the nature of its malfunctioning can result in the onset of a diseased state.

[0114] Accordingly, in diagnosing, preventing, and/or curing such diseased states, determining the genetic makeup of a subject may be extremely useful. For instance, once known, a person's genetic makeup, e.g., his or her genomic composition, can be used for purposes of diagnostics and/or for determining whether a person has or has the potential for a diseased state, and therefore, may be used for prophylaxis. Likewise, the knowledge of a person's genome may be useful in determining various potential therapeutic modalities, such as drugs, that can or cannot be used in a prophylactic or therapeutic regimen without causing harm to the user. In various instances, knowledge of a person's genome may also be employed to determine drug efficacy and/or problematic side effects of such drug use may be predicted and/or identified. Potentially, the knowledge of a person's genome can be used to produce designer drugs, such as drugs tailor made and optimized in accordance with a person's specific genetic makeup. Or, in other instances, such genetic information may be used to determine whether a given subject should be included or not included in a given FDA study. In particular, in one instance, an engineered protein or nucleotide sequence can be fabricated to an individual's unique genetic characteristics so as to turn off or turn on the transcription of genes that either over or under produce proteins and thereby ameliorate diseased states.

[0115] Hence, in some instances, it is a goal of genomics processing to determine individual genomes of people, which determinations may be used in gene discovery protocols as well as for prophylaxis and/or therapeutic regimes to better enhance the livelihood of each particular person and human kind as a whole. Further, knowledge of an individual's genome may be used such as in drug discovery and/or FDA trials to better predict with particularity which, if any, drugs will be likely to work on an individual and/or which would be likely to have deleterious side effects, such as by analyzing the individual's genome and/or a protein profile derived therefrom and comparing the same with a predicted biological response from such drug administration.

[0116] Such genomics and bioinformatics processing usually involves three well defined, but typically separate phases of information processing. The first phase involves DNA/RNA sequencing, where a subject's DNA/RNA is obtained and subjected to various processes whereby the subject's genetic code is converted to a machine-readable digital code, e.g., a FASTQ file. The second phase involves using the subject's generated digital genetic code for the determination of the individual's genetic makeup, e.g., determining the individual's genomic nucleotide sequence and/or variant call file, e.g., how the individual's genome differs from that of one or more reference genomes. And the third phase involves performing one or more analyses on the subject's genetic makeup so as to determine therapeutically useful information therefrom. Sequentially, these may be termed: primary, secondary, and tertiary processing, respectively.

[0117] Preliminarily, e.g., in Phase I, or primary processing, the genetic material must be pre-processed, e.g., via nucleotide sequencing, so as to derive usable genetic sequence data, e.g., a FASTQ file. Accordingly, the sequencing of nucleic acids, such as deoxyribonucleic acid (DNA) and ribonucleic acid (RNA), is a fundamental part of biological discovery. Such detection is useful for a variety of purposes and is often used in scientific research as well as medical advancement. For instance, the genomics and bioinformatics fields are concerned with the application of information technology and computer science to the fields of genetics and/or molecular biology. In particular, genomics and bioinformatics techniques, such as those described herein, can be applied to generate, process, and analyze various genomic data, such as from an individual so as to determine qualitative and quantitative information about that data that can then be used by various practitioners in the development of individual and/or global diagnostic, prophylactic, and/or therapeutic methods for detecting, preventing and/or at least ameliorating diseased states, and thus, improving the safety, quality, and effectiveness of health care for the individual and/or the community.

[0118] Generally, the approach to DNA/RNA analysis, such as for genetic diagnostics, involves nucleic acid hybridization and detection. For example, various typical hybridization and detection approaches include the following steps. For genetic analysis, an RNA or DNA sample of a subject to be analyzed may be isolated and immobilized on a substrate, a probe of a known genetic sequence, e.g., a disease marker, may be labeled and washed across the substrate. If the disease marker is present, a binding event will occur, e.g., hybridization, and because the probe has been labeled the hybridization event may either be or not be detected thereby indicating the presence or absence of the disease marker in the subject's sample. Alternatively, as indicated above, where the hybridization reaction takes place next to a reaction layer, e.g., configured to detect a reactant and/or a by product of the reaction, such as in a suitably configured FET device, a labeled probe need not be employed.

[0119] Typically, for nucleotide sequencing, first, an unknown nucleic acid sequence to be identified, e.g., a single-stranded sequence of DNA and/or RNA of a subject, is isolated, amplified, and immobilized on a substrate. Next, a known nucleic acid labeled with an identifiable tag is contacted with the unknown nucleic acid sequence in the presence of a polymerase. When hybridization occurs, the labeled nucleic acid binds to its complementary base in the unknown sequence immobilized on the surface of the substrate. The binding event can then be detected, e.g., optically or electrically. These steps are then repeated until the entire DNA sample has been completely sequenced. Again, as indicated above, where the complimentary binding event occurs next to reaction layer, e.g., configured to detect a reactant and/or a by product of the binding event, such as in a suitably configured FET device, a labeled nucleic acid need not be employed.

[0120] Generally, these steps are performed manually or via an automated sequencer, such as a Next Gen Sequencer (NGS), wherein thousands to millions of sequences may concurrently be synthesized in the next-generation sequencing process. However, as presented herein, a direct, label-free system for the sequencing of DNA and/or RNA such as on a computer chip, such as a complementary metaloxide semiconductor (CMOS) chip, is presented, such as where various components or the entire sensory apparatus of the sequencer may be embodied within or otherwise associated with the semiconductor chip. Such a system, as herein provided, allows for the seamless integration of primary, secondary, and/or tertiary processing, such as within the same semiconductor chip set.

[0121] More particularly, a typical sequencing procedure, regardless of the type of sequencing apparatus employed, involves obtaining a biological sample from a subject, such as through venipuncture, hair, etc. and treating the sample to isolate the genetic content therefrom. Once isolated, where the genetic sample is DNA, the DNA may be denatured and strand separated. As RNA is already single stranded this step may not be necessary when processing RNA. The isolated DNA and/or RNA or portions thereof may then be multiplied, e.g., via polymerase chain reaction (PCR), so as to build a library of replicated strands that are now ready to be sequenced and read, such as by an automated sequencer, which sequencer is configured to read the replicated strands, e.g., by synthesis, and thereby determine the nucleotide sequences that makes up the DNA and/or RNA. Further, in various instances, such as in building the library of replicated and multiplied strands, it may be useful to provide for over-coverage when preprocessing a given portion of the DNA and/or RNA. To perform this over-coverage, e.g., using PCR, may require increased sample preparation resources and time, and therefore be more expensive, but it often gives an enhanced probability of the end result being more accurate.

[0122] Once the library of replicated DNA/RNA strands has been generated they may be injected into an automated sequencer, e.g., NGS, which may then read the strands, such as by synthesis, so as to determine the nucleotide sequences thereof. For instance, the replicated single stranded DNA or RNA may be attached to a glass bead and inserted into a test vessel, e.g., an array. All the necessary components for replicating its complementary strand, including labeled nucleotides, are also added to the vessel but in a sequential fashion. For example, all "A", "C", "G", and "T's," which may be labeled, are added, either one at a time, or all together, if labeled, to see which of the nucleotides is going to bind at position one of the single stranded DNA or RNA.

[0123] After each addition, in the labeled model, a light, e.g., a laser, is shone on the array. If the composition fluoresces then an image is produced indicating which nucleotide bound to the subject location. In the unlabeled model, a binding event can be detected such as by a change in resistance at a gate, e.g., a solution gate, proximate a reaction layer where the replicated single stranded DNA or RNA containing glass bead is positioned. More particularly, where the nucleotides are added one at a time, if a binding event occurs, then its indicative fluorescence or change in resistance will be observed. If a binding event does not occur, the test vessel may be washed and the procedure repeated until the appropriate one of the four nucleotides binds to its complement at the subject location, and its indicative change in conditions is observed. Where all four nucleotides are added at the same time, each may be labeled with a different fluorescent indicator, and the nucleotide that binds to its complement at the subject position may be determined, such as by the color of its fluorescence. This greatly accelerates the synthesis process.

[0124] Once a binding event has occurred, the complex is then washed and the synthesis steps are repeated for position two. For example, a labeled or otherwise marked nucleotide "A" may be added to the reaction mixture to determine if the complement at position one in the bound template molecule being sequenced is an "A", and if so, the labeled "A" reactant will bind to the template sequence having that complement and will therefore fluoresce, after which the samples will all be washed so as to clear away any excess nucleotide reactants. Where a binding event happened the bound nucleotide is not washed away. This process will be repeated for all nucleotides for all positions until all the over-sampled nucleic acid segments, e.g., reads, have been sequenced and the data collected. Alternatively, where all four nucleotides are added at the same time, each labeled with a different fluorescent indicator, only one nucleotide will bind to its complement at the subject position, and the others will be washed away, such that after the vessel has been washed, a laser may be shone on the vessel and which nucleotide bound to its complement may be determined, such as by the color of its fluorescence. However, where a CMOS FET sensor is employed, as described below, the binding event may be detected by a change in conductance that takes place proximate a suitably configured gate or other reaction region.

[0125] Particularly, in part, due to the need for the use of optically detectable, e.g., fluorescent, labels in the sequencing reactions being performed, the required instrumentation for performing such high throughput sequencing may have a tendency to be bulky, costly, time-consuming, and non-portable. For this reason, a new approach for direct, label-free detection of DNA and/or RNA sequencing are herein proposed. For instance, although in various embodiments, improved methods for performing NGS processing is provided, in other embodiments, improved methods and devices for nucleic acid sequencing and/or processing not necessarily involving an NGS are provided. For example, in particular instances, a detection method is herein proposed that is based on the use of various electronic analytical devices. Such direct electronic detection methods have several advantages over a typical NGS platform.

[0126] More particularly, the sensor and/or detection apparatus, as herein disclosed, may be incorporated in the substrate itself, such as employing a biosystem-on-a-chip device, such as a complementary metal oxide semiconductor device, "CMOS". Specifically, in using a CMOS device in genetic detection, the output signal representative of a hybridization event, e.g., either for hybridization and/or nucleic acid sequencing, can be directly acquired and processed on the microchip itself. In such an instance, automatic recognition is achievable in real time and at a lower cost than is currently achievable using typical NGS processing. Moreover, standard CMOS substrate devices may be employed for such electronic detection making the process simple, inexpensive, rapid, and portable.

[0127] For instance, in order for next-generation sequencing to become widely used as a diagnostic in the healthcare industry, sequencing instrumentation will need to be mass produced with a high degree of quality, mobility, and economy. One way to achieve this is to recast DNA/RNA sequencing in a format that fully leverages the manufacturing base created for computer chips, such as complementary metaloxide semiconductor (CMOS) chip fabrication, which is the current pinnacle of large scale, high quality, low-cost manufacturing of high technology. To achieve this, ideally the entire sensory apparatus of the sequencer may be embodied in a standard semiconductor chip, such as manufactured in the same fab facilities used for logic and memory chips.

[0128] Accordingly, in another aspect of the disclosure, herein presented is a field effect transistor (FET) that may be fabricated on or otherwise associated with a CMOS chip that is configured for use in performing one or more of a DNA/RNA sequencing and/or hybridization reactions. Such a FET may include a gate, a channel region connecting a source and drain terminals, and an insulating barrier that may be configured to separate the gate from the channel. The optimal operation of such a FET relies on the control of the channel conductivity, and thus the control of the drain current, such as by a voltage that may be applied between the gate and source terminals.

[0129] For high-speed applications, and for the purposes of increasing sensor sensitivity, the FETs herein provided can be operated in a manner to respond quickly to variations in the gate voltage (V_{GS}). However, this requires short gates and fast carriers in the channel. In view of this, the present FET sensors, such as for use in nucleic acid hybridization and/or sequencing reactions, are configured so as to have channels that may be very thin in the vertical and/or horizontal dimensions so as to allow for high-speed transmission of carriers as well as for increased sensor sensitivity and accuracy, thereby giving the present sensors particular advantages for nucleic acid sequencing reactions. Therefore, the devices, systems, and methods of employing the same provided herein are ideal for the performance of genomics analysis and applications, such as for nucleic acid sequencing and/or genetic diagnostics.

[0130] Hence, one aspect of the present disclosure is a chemically-sensitive transistor, such as a field effect transistor (FET) that is designed for analysis of biological or chemical materials that solves many of the current problems associated with nucleic acid sequencing and genetic diagnostics. Such FETs may be fabricated on a primary structure, such as a wafer, e.g., a silicon wafer. In various instances, the primary structure may include one or more additional structures, e.g., a second and/or third structure, for instance, in a stacked configuration, such as an insulator material layer. For example, an insulator material may be included on top of the primary structure, and may be an inorganic material, such as a silicon oxide, e.g., a silicon dioxide, or a silicon nitride, or an organic material, such as a polyimide, BCB, or other like material.

[0131] The primary and secondary structures, e.g., including an insulator layer, may include a further structure containing one or more of a conductive source and/or a conductive drain, such as separated one from another by a space, and embedded in the primary structure and/or insulator material and/or may be planar with a top surface of the insulator. In various instances, the structures may further include or may be otherwise associated with a processor, such as for processing generated data, such as sensor derived data. Accordingly, the structures may be configured as, or otherwise include, an integrated circuit, such as herein described, and/or may be an ASIC, a structured ASIC, or an FPGA.

[0132] In particular instances, the structures may be configured as a complementary metal-oxide semiconductor (CMOS), which in turn may be configured as a chemically-sensitive FET containing one or more of a conductive source, a conductive drain, a channel or well, and/or a processor. For instance, the FET may include a CMOS structure having an integrated circuit that is fabricated on a silicon wafer, which further includes an insulator layer, which insulator layer includes the conductive source and the conductive drain, such as embedded therein, which source and drain terminals may be composed of metal, such as a damascene copper source and a damascene copper drain. In various instances, the structures may include a surface, e.g., a top surface, which surface may include a channel, such as where the surface and/or channel may be configured to extend from the conductive source to the conductive drain and form a reaction zone thereby.

[0133] In certain instances, the surface and/or channel may include a one-dimensional transistor material, a two-dimensional transistor material, a three-dimensional transistor material, and/or the like. In various instances, a one-dimensional (1D) transistor material may be included, which 1D material may be composed of a carbon nanotube or a semiconductor nanowire. In other instances, the chamber and/or channel is composed of a one-dimensional transistor material such as containing one or more carbon nanotube(s) and/or a semiconductor nanowire(s), such as a sheet of semiconductor nanowire.

[0134] In particular instances, a two-dimensional (2D) transistor material may be included, such as where the 2D material may be one or two atoms thick and may stretch out in a plane. In such instances, the 2D material may include or otherwise be composed of as elemental 2D materials like graphene, graphyne (a carbon allotrope comprised of a lattice of benzene rings connected by acetylene bonds), borophene (a boron allotrope), germanene (a germanium allotrope), germanane (another germanium allotrope), silicene (a silicon allotrope) stanene (a tin allotrope), phosphorene (a phosphorous allotrope sometimes referred to as black phosphorous) or single atom layers of metals such as palladium or rhodium; a transition metal dichalcogenides (that contain one transition metal atom for every two chalcogen atoms) such as molybdenum disulfide (MoS2 sometimes referred to as molybdenite), tungsten diselenide (WSe2), tungsten disulfide (WS2), or others; MXenes (transition metal carbides and/or nitrides typically of a formula of Mn+1Xn where M is a transition metal and X is carbon and/or nitrogen) such as Ti2C, V2C, Nb2C, Ti3C2, Ti3CN, Nb4C3 or Ta4C3 (furthermore MXenes may be terminated by O, OH or F to produce semiconductors with a small band gap.); or organo-metallic compounds such as Ni HITP (Ni3(2,3,6,7,10,11-hexaiminotriphenylene)2; or 2D supracrystals (the supracrystals are defined as the supra atomic periodic structures where the atoms typically found in the nodes of a structure are replaced by their symmetric complexes. It should be noted that transition metal dichalcogenides may comprise in ratio one atom of any transition metal (Sc, Ti, V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Y, Zr, Nb, Mo, Tc, Ru, Rh, Pd, Ag, Cd, Hf, Ta, W, Re, Os, Ir, Pt, Au, Hg, Rt, Db, Sg, Bh, Mt, Ds or Rg) paired with two atoms of any of the chalcogenides (S, Se or Te). In particular instances, the 2D material may include one or more of a graphene layer, silicene, molybdenum disulfide, black phosphorous, and/or metal dichalcogenides. In various instances, a three-dimensional (3D) material may be included on the surface and/or channel may include a dielectric layer.

[0135] Additionally, in various instances, a reaction layer, e.g., an oxide layer, may be disposed on the surface and/or channel, such as layered or otherwise deposited on the 1D, 2D, e.g., graphene, or 3D layer. Such an oxide layer may be an aluminum oxide or a silicon oxide, such as silicon dioxide. In various instances, a passivation layer may be disposed on the surface and/or channel, such as layered or otherwise deposited on the 1D, 2D, e.g., graphene, or 3D layer and/or on an associated reaction layer on the surface and/or channel.

[0136] In particular instances, the primary and/or secondary and/or tertiary structures may be fabricated or otherwise configured so as to include a chamber or well structure in and/or on the surface. For instance, a well structure may be positioned on a portion of a surface, e.g., an exterior surface, of the primary and/or secondary structures. In some instances, the well structure may be formed on top of, or may otherwise include, at least a portion of the 1D, 2D, e.g., graphene, and/or 3D material, and/or may additionally include the reaction, e.g., oxide, and/or passivation layers. In various instances, the chamber and/or well structure may define an opening, such as an opening that allows access to an interior of the chamber, such as allowing direct contact with the 1D, e.g., carbon nanotube or nanowire, 2D, e.g., graphene, surface and/or channel.

[0137] Accordingly, in various embodiments the present disclosure is directed to a bio-sensor. The bio-sensor includes a CMOS structure that may include a metal containing source, e.g., a damascene copper source, as well as a metal containing drain, e.g., a damascene copper drain, a 1D or 2D layered, e.g., a graphene layered, surface or channel extending from the source to the drain terminals, and a well or chamber structure that may be positioned on a portion of an exterior surface of the 1D or 2D or 3D layered well structure. In such an instance, the well structure may be configured so as to define an opening that allows for direct contact with the nanotube, nanowire, and/or graphene well or chamber surface. In various instances, an oxide and/or passivation layer may be disposed in or on the chamber surfaces. Hence, in certain instances, a chemically-sensitive transistor, such as a field effect transistor (FET) including one or more nano- or micro-wells may be provided.

[0138] In some embodiments, the chemically-sensitive field effect transistor may include a plurality of wells and may be configured as an array, e.g., a sensor array. As such, the system may include an array of wells including one or more, e.g., a plurality, of sensors, such as where each of the sensors includes a chemically-sensitive field-effect transistor having a conductive source, a conductive drain, and a reaction surface or channel extending from the conductive source to the conductive drain. Such an array or arrays may be employed such as to detect a presence and/or concentration change of various analyte types in a wide variety of chemical and/or biological processes, including DNA/RNA hybridization and/or sequencing reactions. For instance, the devices herein described and/or systems including the same may be employed in a method for the diagnosis of disease and/or analysis of biological or chemical materials, such as for whole genome analysis, genome typing analysis, microarray analysis, panels analysis, exome analysis, micro-biome analysis, and/or clinical analysis, such as cancer analysis, NIPT analysis, and/or UCS analysis.

[0139] In a particular embodiment, the FET may be a graphene FET (gFET) array, as herein described, and may be employed to facilitate DNA/RNA sequencing and/or hybridization techniques, such as based on monitoring changes in hydrogen ion concentration (pH), changes in other analyte concentrations, and/or binding events associated with chemical processes relating to DNA/RNA synthesis, such as within a solution gated reaction chamber or well of the gFET based sensor. For example, the chemically-sensitive field effect transistor may be configured as a CMOS biosensor and/or may be adapted to increase the measurement sensitivity and/or accuracy of the sensor and/or associated array (s), such as by including one or more surfaces or wells having a surface layered with a 1D and/or 2D and/or 3D material, a dielectric or reaction layer, a passivation layer and/or the like. For instance, in a particular embodiment, a chemically-sensitive graphene field effect transistor (gFET), such as a gFET having a CMOS structure is provided, where the gFET sensor, e.g., biosensor, may include an oxide and/or passivation layer, such as a layer that is disposed on the surface of the well or chamber so as to increase the measurement sensitivity and/or accuracy of the sensor and/or associated array(s). The oxide layer, when present, may be composed of an aluminum oxide, a silicon oxide, a silicon dioxide, and the like.

[0140] The system may further include one or more of a fluidic component, such as for performing the reaction, a circuitry component, such as for running the reaction processes, and/or a computing component, such as for controlling and/or processing the same. For instance, a fluidics component may be included where the fluidic component is configured to control one or more flows of reagents over the array and/or one or more chambers thereof. Particularly, in various embodiments, the system includes a plurality of reaction locations, such as surfaces or wells, which in turn includes a plurality of sensors and/or a plurality of channels, and further includes one or more fluid sources containing a fluid having a plurality of reagents and/or analytes for delivery to the one or more surfaces and/or wells for the performance of one or more reactions therein. In certain instances, a mechanism for generating one or more electric and/or magnetic fields may also be included.

[0141] The system may additionally include a circuitry component, such as where the circuitry component may include a sample and hold circuit, an address decoder, a bias circuitry, and/or at least one analog-to-digital converter. For instance, the sample and hold circuit may be configured to hold an analog value of a voltage to be applied to or on a selected column and/or row line of an array of a device of the disclosure, such as during a read interval. Additionally, the address decoder may be configured to create column and/or row select signals for a column and/or row of the array, so as to access a sensor with a given address within the array. The bias circuitry may be coupled to one or more surfaces and/or chambers of the array and include a biasing component such as may be adapted to apply a read and/or bias voltage to selected chemically-sensitive field-effect transistors of the array, e.g., to a gate terminal of the transistor. The analog to digital converter may be configured to convert an analog value to a digital value.

[0142] A computing component may also be included, such as where the computing component may include one or more processors, such as a signal processor; a base calling module, configured for determining one or more bases of one or more reads of a sequenced nucleic acid; a mapping module, configured for generating one or more seeds from the one or more reads of sequenced data and for performing a mapping function on the one or more seeds and/or reads; an alignment module, configured for performing an alignment function on the one or more mapped reads; a sorting module, configured for performing a sorting function on the one or more mapped and/or aligned reads; and/or a variant calling module, configured for performing a variant call function on the one or more mapped, aligned, and/or sorted reads. In particular instances, the base caller of the base calling module may be configured to correct a plurality of signals, such as for phase and signal loss, to normalize to a key, and/or to generate a plurality of corrected base calls for each flow in each sensor, so as to produce a plurality of sequencing reads. In various embodiments, the device and/or system may include at least one reference electrode.

[0143] Particularly, the system may be configured for performing a sequencing reaction. In such an instance, the chemically-sensitive FET sequencing device may include an array of sensors having one or more chemically-sensitive field-effect transistors associated therewith. Such transistors may include a cascode transistor having one or more of a source terminal, a drain terminal, and or a gate terminal. In such an instance, the source terminal of the transistor may be directly or indirectly connected to the drain terminal of the field-effect transistor. In some instances, a one or two dimensional channel may be included and may extend from the source terminal to the drain terminal, such as where the 1D channel material may be a carbon nanotube or nanowire, and the two-dimensional channel material may be composed of graphene, silicene, a phosphorene, a molybdenum disulfide, and a metal dichalcogenide. The device may further be configured to include a plurality of column and row lines coupled to the sensors in the array of sensors. In such an instance, each column line in the plurality of column lines may be directly or indirectly connected to or otherwise coupled to the drain terminals of the transistors, e.g., cascode transistors, of a corresponding plurality of signals or pixels in the array, and likewise each row line in the plurality of row lines may be directly or indirectly connected to or otherwise coupled with the source terminals of the transistors, e.g., cascode transistors, of a corresponding plurality of sensors in the array.

[0144] In some instances, a plurality of source and drain terminals having a plurality of reaction surfaces, e.g., channel members, extended there between may be included, such as where each channel member includes a one or two or three dimensional material. In such an instance, a plurality of first and/or second conductive layers may be coupled to the first and second source/drain terminals of the chemically-sensitive field-effect transistors in respective columns and rows in the array. Additionally, control circuitry may be provided and coupled to the plurality of column and row lines such as for reading a selected sensor connected to a selected column line and/or a selected row line. The circuitry may also include a biasing component such as may be configured for applying a read voltage to the selected row line, and/or to apply a bias voltage such as to the gate terminal of a transistor, such as FET and/or cascode transistor of the selected sensor. In a particular embodiment, the bias circuitry may be coupled to one or more chambers of the array and be configured to apply a read bias to selected chemically-sensitive field-effect transistors associated with the chamber via the conductive column and/or row lines. Particularly, the bias circuitry may be configured to apply a read voltage to the selected row line, and/or to apply a bias voltage to the gate terminal of the transistor, e.g., cascode transistor, such as during a read interval.

[0145] A sense circuitry may be included and coupled to the array so as to sense a charge coupled to one or more of the gate configurations of a selected chemically-sensitive field-effect transistor. Sense circuitry may also be configured to read the selected sensor based on a sampled voltage level on the selected row and/or column line. In such an instance, the sense circuitry may include one or more of a pre-charge circuit, such as to pre-charge the selected column line to a pre-charge voltage level prior to the read interval; and a sample circuit such as to sample a voltage level at the drain terminal of the selected transistor, e.g., cascode transistor, such as during the read interval. The sample circuit may also be included and contain a sample and hold circuit configured to hold an analog value of a voltage on the selected column line during the read interval, and may further include an analog to digital converter to convert the analog value to a digital value.

[0146] In another aspect, the present 1D, 2D, or 3D FET integrated circuits, e.g., a gFET, sensors, and/or arrays of the disclosure may be fabricated such as using any suitable complementary metal-oxide semiconductor (CMOS) processing techniques known in the art. In certain instances, such a CMOS processing technique may be configured to increase the measurement sensitivity and/or accuracy of the sensor and/or array, and at the same time facilitate significantly small sensor sizes and dense gFET chamber sensor regions. Particularly, the improved fabrication techniques herein described employing a 1D, 2D, 3D, and/or oxide as a reaction layer provide for rapid data acquisition from small sensors to large and dense arrays of sensors. In particular embodiments, where an ion-selective permeable membrane is included, the membrane layer may include a polymer, such as a perfluorosulphonic material, a perfluorocarboxylic material, PEEK, PBI, Nafion, and/or PTFE. In some embodiments, the ion-selective permeable membrane may include an inorganic material, such as an oxide or a glass. One or more of the various layers, e.g., the reaction, passivation, and/or permeable membrane layers may be fabricated or otherwise applied by a spin-coating, anodization, PVD, and/or sol gel method.

[0147] Accordingly, the CMOS FET device described herein may be employed for sequencing a nucleic acid sample, in such an instance the nucleic acid sample serves as a template for DNA/RNA synthesis and sequencing that may be coupled to or in proximity with the surface, e.g., a graphene coated surface, of the reaction zone, such as within a suitably configured well or chamber of the CMOS FET sensor array. Once immobilized the template sequence may then be sequenced and/or analyzed by performing one or more of the following steps. For example, a primer, and/or a polymerase, e.g., a DNA and/or RNA polymerase, and/or one or more substrates, e.g., a deoxynucleotide triphosphate, such as dATP, dGTP, dCTP, and dTTP (or dUTP for RNA) or di-deoxynucleotide triphosphate, such as ddATP, ddGTP, ddCTP, and ddTTP/ddUTP, may be added, e.g., sequentially, to the reaction chamber, such as after the hybridization reaction begins so as to induce an elongation reaction. Once the appropriate, e.g., corresponding, substrate hybridizes to its complement in the template sequence, there will be a concomitant change in the individual electrical characteristic voltage, e.g., the source-drain voltage (Vsd), measured as a result of the new local gating effect. Where a reaction layer is included, such as an oxide layer deposited upon the 1D, 2-D, or 3-D surface, the sensitivity with which a binding event occurs can be amplified, such as where the reaction layer is configured for producing and/or monitoring changes in hydrogen ion concentration (pH), or changes in other analyte concentrations.

[0148] Hence, for every elongation reaction with the appropriate, e.g., complementary, substrate there will be a change in the characteristic voltage and/or pH concentration. For instance, as described herein, a field-effect device for nucleic acid sequencing and/or gene detection may disposed in a sample chamber or well of a flow cell, and a sample solution, e.g., containing a polymerase and one or more substrates, e.g., nucleotides, may be introduced to the sample solution chamber, such as via one or more of the fluidics components of the system. In various embodiments, a reference electrode may be disposed upstream, downstream, and/or in fluid contact with the field effect device and/or the source and/or drain terminals may themselves serve as electrodes, such as for hybridization detection, and gate voltage may be applied whenever needed.

[0149] Particularly, in an exemplary elongation reaction, such as described above, polynucleotides are synthesized, into a complementary nucleotide strand, if the added substrate is complementary to the next available base sequence of the target DNA/RNA template and in the presence of a primer. If the added substrate is not complementary to the next available base sequence in the template, hybridization does not occur and there is no elongation. Since nucleic acids, such as DNAs and RNAs, have a negative charge in aqueous solutions, hybridization resulting in elongation can be incrementally determined by the change in the charge density on the reaction surface and/or in the reaction chamber. Such detection may be enhanced by being able to detect increases in ion concentration, such as by detecting a change in the pH. Because the substrates are added sequentially, it can readily be determined which nucleotide bound to the template thereby facilitating the elongation reaction. Accordingly, as a result of elongation, the negative charge on the graphene layered gate surface, insulating film surface, and/or the sidewall surface of the reaction chamber will be increased. This increase may then be detected, such as a change in the gate source voltage and/or ion concentration, as described in detail herein. By determining the addition of which substrate resulted in a signal or pH change in gate-source voltage, the base sequence identity of the target nucleic acid can be determined and/or analyzed.

[0150] Particularly, regardless of the sequencing device employed, such as an NGS and/or a FET based sequencing device, as herein described, this iterative synthesis process continues until the entire DNA/RNA template strand has been replicated in the vessel. Usually a typical length of a sequence replicated in this manner is from about 100 to about 500 base pairs, such as between 150 to about 400 base pairs, including from about 200 to about 350 base pairs, such as about 250 base pairs to about 300 base pairs dependent on the sequencing protocol being employed. Further, the nucleotide length of these template segments may be pre-determined, e.g., engineered, to accord with any particular sequencing machinery and/or protocol by which it is run.

[0151] The end result is a readout, or read, that is comprised of a replicated DNA/RNA segment, e.g., from about 100 to about 1,000 nucleotides or more in length, that has either been labeled in such a manner that every nucleotide in the sequence, e.g., read, is known because of its label, or is determined and known by a change in a gate characteristic, such as a change in voltage and/or pH, e.g., at a gate region in the sequencing sensor device. Hence, since the human genome is comprised of about 3.2 billion base pairs, and various known sequencing protocols usually result in detectable replicated sequences, e.g., reads, from about 100 or 101 bases to about 250 or about 300 or about 400 bases, the total amount of segments that need to be sequenced, and consequently the total number of reads generated, can be anywhere from about 10,000,000 to about 40,000,000, such as about 15,000,000 to about 30,000,000, dependent on how long the label replicated sequences are. Therefore, the sequencer may typically generate about 30,000,000 reads, such as where the read length is 100 nucleotides in length, so as to cover the genome once. However, as indicated herein, due to the condensed nature of the present sequencing on a chip format presented herein, much more substantial read lengths, such as 800 bases, 1,000 bases, 2,500 bases, 5,000 bases, up to 10,000 bases may be achievable.

[0152] Further, as indicated above, in such procedures, it may be useful to oversample the DNA/RNA such by about 5×, or about 10×, or about 20×, or about 25×, or about 30×, or about 40×, or about 50×, or about 100×, or about 200×, or about 250×, or about 500×, or about 1,000×, or about 5,000×, or even about 10,000× or more, and as such the amount of primary processing needed to be done and the time taken to do this can be quite extensive. For instance, with 40× oversampling, wherein the various synthesized reads are designed to overlap to some extent, up to about 1.2 billion reads may need to be synthesized. Typically, a large majority if not all of these detectable sequences can be generated in parallel. The end result is that the initial biological genetic material is processed, e.g., by amplification and/or sequencing protocols such as those summarized herein, and a digital representation of that data is generated, e.g., a FASTQ file, which digital representation of data may be subjected to a secondary processing protocol.

[0153] Particularly, the genetic material of a subject may be replicated and sequenced in such a manner that a measurable electrical, chemical, radioactive, and/or optical signal is generated, which signal is then detected and converted, e.g., by the sequencer and/or a processing apparatus associated therewith, into a digital representation of the subject's genetic code. More particularly, primary processing may include the conversion of images, such as recorded flashes of light or other electrical or chemical signal data, into FASTQ file data. Accordingly, this information is stored as a FASTQ file, which may then be sent for further, e.g., secondary processing. A typical FASTQ file includes a large collection of reads representing digitally encoded nucleotide sequences wherein each predicted base in the sequence has been called and given a probability score that the called base at the indicated position is incorrect.

[0154] In many instances, it may be useful to further process the digitally encoded sequence data obtained from the sequencer and/or sequencing protocol, such as by subjecting the digitally represented data to secondary processing. This secondary processing, for instance, can be used to assemble an entire genomic profile of an individual, such as where the individual's entire genetic makeup is determined, for instance, where each and every nucleotide of each and every chromosome is determined in sequential order such that the composition of the individual's entire genome has been identified. In such processing, the genome of the individual may be assembled such as by comparison to a reference genome, such as a standard, e.g., one or more genomes obtained from the human genome project, so as to determine how the individual's genetic makeup differs from that of the referent(s). This process is commonly known as variant calling. As the difference between the DNA/RNA of any one person to another is 1 in 1,000 base pairs, such a variant calling process can be very labor and time intensive.

[0155] Accordingly, in a typical secondary processing protocol, a subject's genetic makeup is assembled by comparison to a reference genome. This comparison involves the reconstruction of the individual's genome from millions upon millions of short read sequences and/or the comparison of the whole of the individual's DNA and/or RNA to an exemplary DNA and/or RNA sequence model. In a typical secondary processing protocol a FASTQ file is received from the sequencer containing the raw sequenced read data. For instance, in certain instances, there can be up to 30,000,000 reads or more covering the subject's genome, assuming no oversampling, such as where each read is about 100 nucleotides in length. Hence, in such an instance, in order to compare the subject's DNA/RNA genome to that of the standard reference genome, it needs to be determined where each of these reads map to the reference genome, such as how each is aligned with respect to one another, and/or how each read can also be sorted by chromosome order so as to determine at what position and in which chromosome each read belongs. One or more of these functions may take place prior to performing a variant call function on the entire full-length sequence. Once it is determined where in the genome each read belongs, the full length genetic sequence may be determined, and then the differences between the subject's genetic code and that of the referent can be assessed.

[0156] As the human genome is over 3 billion base pairs in length, efficient automated sequencing protocols and machinery have been developed so as to effectuate the sequencing of such DNA/RNA genomes within a time period that could be clinically useful. Such innovations in automated sequencing have resulted in the capabilities of sequencing an entire genome in a matter of hours to days dependent on the number of genomes being sequenced, the amount of oversampling involved, and the number of processing resources being dedicated to the job. Hence, given these advancements in sequencing, a large amount of sequencing data is capable of being generated in a relatively short period of time. A result of these advancements, however, is the development of a bottleneck at the secondary processing stage. In efforts to help overcome this bottleneck various software-based algorithms, such as those described herein, have been developed to help expedite the process of assembling a subject's sequenced DNA and/or RNA such as by a reference based assembly process.

[0157] For instance, reference based assembly is a typical secondary processing assembly protocol involving the comparison of sequenced genomic DNA and/or RNA of a subject to that of one or more standards, e.g., known reference sequences. Various algorithms have been developed to help expedite this process. These algorithms typically include some variation of one or more of: mapping, aligning, and/or sorting the millions of reads received from the digital, e.g., FASTQ, files communicated by the sequencer, to determine where on each chromosome each particular read corresponds or is otherwise located. Often a common feature behind the functioning of these various algorithms is their use of an index and/or an array to expedite their processing function.

[0158] For instance, with respect to mapping, a large quantity, e.g., all, of the sequenced reads may be processed to determine the possible locations in the reference genome to which those reads could possibly align. One methodology that can be used for this purpose is to do a direct comparison of the read to the reference genome so as to find all the positions of matching. Another methodology is to employ a prefix or suffix array, or to build out a prefix or suffix tree, for the purpose of mapping the reads to various positions in the reference DNA/RNA genome. A typical algorithm useful in performing such a function is a Burrows-Wheeler transform, which is used to map a selection of reads to a reference using a compression formula that compresses repeating sequences of data.

[0159] A further methodology is to employ a hash table, such as where a selected subset of the reads, a k-mer of a selected length "k", e.g., a seed, are placed in a hash table as keys and the reference sequence is broken into equivalent k-mer portions and those portions and their location are inserted by an algorithm into the hash table at those locations in the table to which they map according to a hashing function. A typical algorithm for performing this function is "BLAST", a Basic Local Alignment Search Tool. Such hash table based programs compare query nucleotide or protein sequences to one or more standard reference sequence databases and calculates the statistical significance of matches. In such manners as these, it may be determined where any given read is possibly located with respect to a reference genome. These algorithms are useful because they require less memory, fewer look ups, and therefore require fewer processing resources and time in the performance of their functions, than would otherwise be the case, such as if the subject's genome were being assembled by direct comparison, such as without the use of these algorithms.

[0160] Additionally, an aligning function may be performed to determine out of all the possible locations a given read may map to on a genome, such as in those instances where a read may map to multiple positions in the genome, which is in fact the location to which it actually was derived, such as by being sequenced therefrom by the original sequencing protocol. This function may be performed on a number of the reads of the genome and a string of ordered nucleotide bases representing a portion or the entire genetic sequence of the subject's DNA and/or RNA may be obtained. Along with the ordered genetic sequence a score may be given for each nucleotide position, representing the likelihood that for any given nucleotide position, the nucleotide, e.g., "A", "C", "G", "T" (or "U"), predicted to be in that position is in fact the nucleotide that belongs in that assigned position. Typical algorithms for performing alignment functions are Needleman-Wunsch and Smith-Waterman. In either case, these algorithms perform sequence alignments between a string of the subject's query genomic DNA and/or RNA sequence and a string of the reference genomic sequence whereby instead of comparing the entire genomic sequences, one with the other, segments of a selection of possible lengths are compared.

[0161] Once the reads have been assigned a position, such as relative to the reference genome, which may include identifying to which chromosome the read belongs and/or its offset from the beginning of that chromosome, the reads may be sorted by position. This may enable downstream analyses to take advantage of the oversampling described above. All of the reads that overlap a given position in the genome will be adjacent to each other after sorting and they can be organized into a pileup and readily examined to determine if the majority of them agree with the reference value or not. If they do not, a variant can be flagged.

[0162] Although these algorithms and the others like them go a ways to resolving the bottlenecks inherent in secondary processing, faster performance time and better accuracy are still desirable. More particularly, although there has been advancement in the generation of raw data, such as generated DNA/RNA sequence data, the advancements in information technologies have not kept up pace, leading to a data analysis bottleneck. This bottleneck is somewhat lessened by the development of various algorithms, such as those described above, which help accelerate these analyses, but there still exists a need for new technologies to handle the data generation and acquisition, computation, storage, and/or analysis of such data, especially as it relates to genomic sequence analysis, such as in a secondary processing stage.

[0163] For instance, employing standard NGS technologies it can take several hours, up to about a day, to sequence a human genome, and using standard protocols for performing secondary processing on such obtained genomic sequencing data, can take up to three (3) days or even up to a week or more to process the sequenced data so as to generate clinically relevant genomic sequence information of an individual. Employing various different optimized devices, algorithms, methods, and/or systems the time expended for primary to secondary processing can be brought down to a mere 27 to 48 hours. However, in order to achieve such rapid results typically requires virtually all the generated reads, e.g., 30 million reads of 100 nucleotides each, to be processed in parallel and at the same time. Such parallel processing requires extensive processing power involving massive CPU and other energy consuming resources and still takes a relatively long time.

[0164] Further, in various instances, enhanced accuracy of results is desired. Such enhanced accuracy can be achieved through providing some amount of oversampling of the sequenced genome. For example, as described above, it may be desirable to process the subject's DNA/RNA in such a manner that at any given location of a sequence of nucleotides, there is an oversampling of that region. As indicated above, it may be desired to oversample any given region of the genome up to 10×, or 15×, or 20×, or 25×, or 30×, or 40×, 50×, 100×, 250× or even 500× or 1,000 times or more. However, where the genome is oversampled, such as by 40×, the amount of reads to be processed is roughly 30 Million× 40 (dependent on the length of the reads), which amounts to about 1.2 billion reads that need to be processed, when the entire genome is oversampled by 40×. Hence, although such oversampling typically results in greater accuracy, it is at a cost of taking more time and requiring more extensive processing resources as each section of the genome is covered by anywhere from 1 to 40 times. Moreover, for certain oncology applications in which a clinician is trying to distinguish between the mutated genome of cancer cells in the blood stream as distinct from the genome of healthy cells, oversampling of as much as 500×, or 1,000×, or 5,000×, or even 10,000× may be employed.

[0165] The present disclosure, therefore, is directed to such new technologies that may be implemented in one or a series of genomics and/or bioinformatics protocols, e.g., pipelines, for performing genetic acquisition and/or analysis, such as primary and/or secondary processing, on obtained genomic sequencing data or a portion thereof. The sequencing data may be obtained directly from an automated high throughput sequencer system, such as by a "Sequencing by Synthesis" 454 automated sequencer from ROCHE, a HiSeq×Ten or a Solexia automated sequencer from ILLUMINA, a "Sequencing by Oligonucleotide Ligation and Detection" (SOLiD) or Ion Torrent sequencer by LIFE TECHNOLOGIES, and/or a "Single Molecule Fluorescent Sequencing" sequencer by HELICOS GENETIC ANALYSIS SYSTEMS, or the like, such as by a direct linkage with the sequencing processing unit, or the sequencing data may be obtained directly such as in a sequencing on a chip configuration, such as a graphene layered FET sensor containing CMOS sequencing chip, as herein described. Such sequencing data may also be obtained remotely, such as from a database, for instance, accessible via the internet or other remote location accessible through a wireless communications protocol, such as Wi-Fi, Bluetooth, or the like.

[0166] In certain aspects, these genetic acquisition and/or analysis technologies may employ improved algorithms that may be implemented by software that is run in a less processing intensive and/or less time consuming manner and/or with greater percentage accuracy. For instance, in certain embodiments, improved devices and methods for producing genetic sequence information, such as in a primary processing protocol, as disclosed herein, and/or improved algorithms for performing secondary processing thereon, as disclosed herein, is provided. In various particular embodiments, the improved devices, systems, their methods of use, and the algorithms employed are directed to more efficiently and/or more accurately performing one or more of sequencing, mapping, aligning, and/or sorting functions, such as to generate and/or analyze a digital representation of DNA/RNA sequence data obtained from a sequencing platform, such as in a FASTQ file format obtained from an automated sequencer and/or sequencer on a chip, such as one of those set forth above.

[0167] Additionally, in certain embodiments, improved algorithms directed to more efficiently and/or more accurately performing one or more of local realignment, duplicate marking, base quality score recalibration, variant calling, compression, and/or decompression functions are provided. Further, as described in greater detail herein below, in certain aspects, these genetic production and/or analysis technologies may employ on or more algorithms, such as improved algorithms, that may be implemented by hardware that is run in a less processing intensive and/or less time consuming manner and/or with greater percentage accuracy than various software implementations for doing the same.

[0168] In particular embodiments, a platform of technologies for sequencing DNA/RNA so as to produce genetic sequence data and/or performing genetic analyses are provided where the platform may include the performance of one or more of: sequencing, mapping, aligning, sorting, local realignment, duplicate marking, base quality score recalibration, variant calling, compression, and/or decompression functions, and/or may further include tertiary processing protocols, as herein described. In certain instances, the implementation of one or more of these platform functions is for the purpose of generating and/or performing one or more of determining and/or reconstructing a subject's consensus genomic sequence, comparing a subject's genomic sequence to a referent sequence, e.g., a reference or model genetic sequence, determining the manner in which the subject's genomic DNA and/or RNA differs from a referent, e.g., variant calling, and/or for performing a tertiary analysis on the subject's genomic sequence, such as for whole genome analysis, such as genome-wide variation analysis and/or genome typing analysis, gene function analysis, protein function analysis, e.g., protein binding analysis, quantitative and/or assembly analysis of genomes and/or transcriptomes, micro-array analysis, panels analysis, exome analysis, micro-biome analysis, and/or clinical analysis, such as cancer analysis, NIPT analysis, and/or UCS analysis, as well as for various diagnostic, and/or a prophylactic and/or therapeutic evaluation analyses.

[0169] Particularly, once the genetic data has been generated and/or processed, e.g., in one or more primary and/or secondary processing protocols, such as by being mapped, aligned, and/or sorted, such as to produce one or more variant call files, for instance, to determine how the genetic sequence data from a subject differs from one or more reference sequences, a further aspect of the disclosure may be directed to performing one or more other analytical functions on the generated and/or processed genetic data such as for further, e.g., tertiary, processing. For example, the system may be configured for further processing of the generated and/or secondarily processed data, such as by running it through one or more tertiary processing pipelines, such as one or more of a genome pipeline, an epigenome pipeline, metagenome pipeline, joint genotyping, a MuTect2 pipeline, or other tertiary processing pipeline, such as by the devices and methods disclosed herein. For instance, in various instances, an additional layer of processing may be provided, such as for disease diagnostics, therapeutic treatment, and/or prophylactic prevention, such as including

NIPT, NICU, Cancer, LDT, AgBio, and other such disease diagnostics, prophylaxis, and/or treatments employing the data generated by one or more of the present primary and/or secondary and/or tertiary pipelines. Hence, the devices and methods herein disclosed may be used to generate genetic sequence data, which data may then be used to generate one or more variant call files and/or other associated data that may further be subject to the execution of other tertiary processing pipelines in accordance with the devices and methods disclosed herein, such as for particular and/or general disease diagnostics as well as for prophylactic and/or therapeutic treatment and/or developmental modalities.

[0170] Further, in various embodiments, a bioinformatics processing regime, as disclosed herein, may be employed for the purpose of creating one or more masks, such as a genome reference mask, a default mask, a disease mask, and/or an iterative feed back mask, which may be added to the mapper and/or aligner, e.g., along with a reference, wherein the mask set is configured so as to identify a particular area or object of interest. For instance, in one embodiment, the methods and apparatuses described herein may be employed so as to create genome reference mask, such as by creating a mask-set that can be loaded into the mapper and/or aligner along with a reference, wherein the mask set is configured so as to identify areas of high importance and/or relevance, e.g., to the practitioner or subject, and/or so as to identify areas having increased susceptibility to errors. In various embodiments, the mask-set may provide intelligent guidance to the mapper and/or aligner such as on which areas of the genome to focus on to improve quality. Masks, therefore, can be created in a layered manner to provide varying levels or iterations of guidance based on various specific applications. Each mask accordingly could identify the areas of interest and provide a minimum quality target for the area. Additionally, a default mask may be employed to provide guidance, such as on an identified, e.g., typical, “high value” areas of the genome. Such areas could include known coding areas, control areas, etc. as well as areas that are well known to produce errors. Further, a disease mask, or application specific mask, may be employed to the mask-set that identifies areas of high importance, such as areas that require very high levels of accuracy based on known markers, e.g., Cancer. Further still, iterative feedback masking may be employed, such as by adding a new, ad-hoc mask, that may be specifically designed by using feedback from a tertiary analysis system (like Cypher Genomics) that has identified areas of concern based on observed errors or inconsistencies.

[0171] As indicated above, in one aspect one or more of these platform functions, e.g., mapping, aligning, sorting, realignment, duplicate marking, base quality score recalibration, variant calling, one or more tertiary processing modules, compression, and/or decompression functions is configured for implementation in software. In another embodiment, one or more of these platform functions, e.g., mapping, aligning, sorting, local realignment, duplicate marking, base quality score recalibration, decompression, variant calling, tertiary processing, compression, and/or decompresion functions is configured for implementation in hardware.

[0172] Accordingly, in certain instances, methods are presented herein where the method involves the performance of an algorithm, such as an algorithm for performing one or more genetic analysis functions such as mapping, aligning, sorting, realignment, duplicate marking, base quality score recalibration, variant calling, compression, and/or decompression where the algorithm has been optimized in accordance with the manner in which it is to be implemented. In particular, where the algorithm is to be implemented in a software solution, the algorithm and/or its attendant processes, has been optimized so as to be performed faster and/or with better accuracy for execution by that media. Likewise, where the functions of the algorithm are to be implemented in a hardware solution, the hardware has been designed to perform these functions and/or their attendant processes in an optimized manner so as to be performed faster and/or with better accuracy for execution by that media. These methods, for instance, can be employed such as in an iterative variant calling procedure.

[0173] Hence, in one aspect, presented herein are systems, apparatuses, and methods for implementing bioinformatic protocols, such as for performing one or more functions for analyzing genetic data, such as genomic data, for instance, via one or more optimized algorithms and/or on one or more optimized integrated circuits, such as on one or more hardware processing platforms. Hence, in one instance, systems and methods are provided for implementing one or more algorithms for the performance of one or more steps for analyzing genomic data in a bioinformatics protocol, such as where the steps may include the performance of one or more of: mapping, aligning, sorting, local realignment, duplicate marking, base quality score recalibration, variant calling, compression, and/or decompression. In another instance, systems and methods are provided for implementing the functions of one or more algorithms for the performance of one or more steps for analyzing genomic data in a bioinformatics protocol, as set forth herein, wherein the functions are implemented on a hardware accelerator, which may or may not be coupled with one or more general purpose processors and/or super computers.

[0174] More specifically, in some instances, methods for performing secondary analytics on data pertaining to the genetic composition of a subject are provided. In one instance, the analytics to be performed may involve reference based reconstruction of the subject genome. For instance, referenced based mapping involves the use of a reference genome, which may be generated from sequencing the genome of a single or multiple individuals, or it may be an amalgamation of various people’s DNA that have been combined in such a manner so as to produce a prototypical, standard reference genome to which any individual’s DNA may be compared, for example, so as to determine and reconstruct the individual’s genetic sequence and/or for determining the difference between their genetic makeup and that of the standard reference, e.g., variant calling.

[0175] More particularly, a reason for performing a secondary analysis on a subject’s sequenced DNA is to determine how the subject’s DNA varies from that of the reference. More specifically, to determine one, a multiplicity, or all the differences in the nucleotide sequence of the subject from that of the reference. For instance, the differences between the genetic sequences of any two random persons is 1 in 1,000 base pairs, which when taken in view of the entire genome of over 3 billion base pairs amounts to a variation of up to 3,000,000 divergent base pairs per person. Determining these differences may be useful such as in a tertiary analysis protocol, for instance, so as to predict the potential for the occurrence of a diseased state, such as because of a genetic abnormality, and/or the likelihood of success of a prophylactic or therapeutic modality, such as based on how a prophylactic or therapeutic is expected to interact with the subject's DNA or the proteins generated therefrom. In various instances, it may be useful to perform both a de novo and a reference based reconstruction of the subject's genome so as to confirm the results of one against the other, and to, where desirable, enhance the accuracy of a variant calling protocol.

[0176] In various instances, as set forth above, it may be useful in performing a primary sequencing protocol to produce oversampling for one or more regions of the subject's genome. These regions may be selected based on known areas of increased variability, suspected regions of variability, such as based on the condition of the subject, and/or on the entire genome generally. In its basic form, as indicated above, based on the type of sequencing protocols performed, sequencing produces readouts, e.g., reads, that are digital representations of the subject's genetic sequence code. These read lengths are typically designed based on the type of sequencing machinery being employed. For instance, the 454 automated sequencer from ROCHE, typically produces read lengths from 100 or 150 base pairs in length to about 1,000 base pairs; for ILLUMINA the read lengths are typically engineered to be from about 100 or 101 to about 150 base pairs in length for some of their technology, and 250 base pairs in length for other of their technology; for LIFE TECHNOLOGIES the read lengths are typically engineered to be from about 50 to about 60 base pairs in length for their SOLiD technology and from 35 to 450 base pairs in length for their Ion Torrent technology; and for the HELICOS GENETIC ANALYSIS SYSTEMS the read lengths may vary but may typically be less than 1,000 nucleotides in length.

[0177] However, because the processing of the DNA sample required to produce engineered read lengths of a specific size is both labor and chemistry intensive, and because the sequencing itself often depends on the functioning of the sequencing machinery, there is some possibility that errors may be made throughout the sequencing process thereby introducing an abnormality into that portion of the sequenced genome where the error occurred. Such errors can be problematic especially where a purpose for reconstructing the subject's genome is to determine how it or at least a portion of the genome varies from a standard or model reference. For instance, a machine or chemistry error resulting in the change of one nucleotide, e.g., in a read, for another will give a false indication of a variation that is not really there. This can result in an incorrect variant call and may further result in the false indication of a diseased state and the like. Accordingly, because of the possibility of machine, chemistry, and/or even human error in the execution of a sequencing protocol, in many instances, it is desirable to build redundancy into an analysis system, such as by oversampling portions of or the entire genome. More particularly, as an automated sequencer produces a FASTQ file calling out a sequence of reads having nucleotides at a given position along with the probability that the call for a given nucleotide being at the called position is actually incorrect, e.g., a base call, it is often desirable to employ methods, such as oversampling, for ensuring that base calls made by the sequencing processes can be detected and corrected.

[0178] Hence, in performing the methods herein described, in certain instances, a primary sequencing protocol is performed in such a manner so as to produce a sequenced genome where a portion or the entire genome is oversampled by about 10×, about 15×, about 20×, about 25×, about 30× about 40×, such as about 50× or more. Accordingly, where the read lengths are engineered to be about 50-60 base pairs in length, this oversampling can result in about 2 to about 2.5 billion reads, or where the read lengths are about 100 or 101 base pairs in length, oversampling may result in about 1 to about 1.2 billion reads, and where the read lengths are about 1,000 base pairs in length, about 50 to about 100 million reads may be generated by the sequencer, such as where the oversampling is about 40×. More particularly, in such an instance, because of the 40× oversampling, at any given point in the genome it is expected that there will be 40 reads to cover any one position albeit, the given position might be at the beginning of one read, the middle of another, and the end of another, but it is expected to be covered about 40 times.

[0179] Therefore, such oversampling produces regions of the sequenced genome that are covered by a multiplicity of reads, e.g., duplications, such as up to about 40 reads, for instance, where the oversampling is about 40×. These at least partial duplications are useful in determining whether any given variation in any particular read is in fact an actual genomic variation or rather a machine or chemistry artifact. Hence, oversampling can be employed to improve the accuracy in reconstructing the subject's genome, especially in instances where the subject's genome is to be compared against a reference genome so as to determine those instances where the subject's genetic sequence differs from that of the reference genetic sequence. In a manner such as this, as described in greater detail herein below, it can be confirmed that any given variation between the reconstructed sequence and the model is in fact due to the presence of an actual variant and not an error in the initial processing of sample DNA, or read alignment software, etc.

[0180] For instance, in building the genetic sequence of the individual's sequenced DNA, it must be determined what nucleotide goes where in the growing string of nucleotides. In order to determine what nucleotide goes where, the various reads can be organized and a pile up of reads covering duplicate locations can be built up. This allows for a comparison to be made of all the reads covering the same locations so as to more accurately determine if there is an actual variation at any given position or if there may be an error in any one read at the position in question in the pileup. For example, if there is only one or two of the reads out of the 40 that has a particular nucleotide at position X, and all 38 or 39 other reads agree on a different nucleotide being at that position, then the two outlying reads may be excluded as being in error, at least at this specific location.

[0181] More particularly, where there are a multiplicity of reads generated for any one location of the subject's genome, there are likely to be multiple overlaps or pile-ups for any given nucleotide position. These pile-ups represent the coverage for any particular location and may be useful for determining with better accuracy the correct sequence of the subject's genome. For instance, as indicated, sequencing results in the production of reads, and in various instances, the reads produced are over sampled, and so at various positions various particular reads will overlap. This overlapping is useful for determining the actual sample genome such as with a high probability of correctness.

[0182] The purpose, therefore, may be to scan over the reference genome incrementally multiple times, as described in greater detail herein below, so as to more accurately reconstruct the subject's genome, and where it is desirable to determine how the subject's genome differs from a different genome, e.g., a model genome, the use of pile-ups can more accurately identify errors, such as chemical, machine, or read errors, and distinguish them from actual variants. More specifically, where the subject has an actual variation at position X, the majority of reads in the pile up should verify, e.g., include, that variation. Statistical analysis procedures, such as those described herein, may then performed to determine the actual genetic sequence of the subject with all its variants from a reference genome.

[0183] For instance, where the subject's genetic sequence is to be rebuilt with respect to the use of a reference genome, once the reads, e.g., a pile-up of reads, have been generated, the next steps may be to map and/or align and/or sort the reads to one or more reference genomes (e.g., the more exemplary reference genomes available as models the better the analysis is likely to be) and thereby rebuild the genome of the subject, this results in a series of reads that have been mapped and/or aligned with the reference genome(s) at all possible positions along the chain where there is a match, and at each such position they are given a probability score as to the probability that they actually belong in that position.

[0184] Accordingly, in various instances, once the reads have been generated, their positions mapped, e.g., the potential locations in the reference genome to which the reads may map have been determined, and their sequential order aligned, the actual genetic sequence of the subject's genome may be determined, such as by performing a sorting function on the aligned data. Further, once the actual sample genome is known and compared to the reference genome, the variations between the two can be determined, a list of all the variations/deviations between the reference genome and the sample genome are determined and called out. Such variations between the two genetic sequences may be due to a number of reasons.

[0185] For instance, there may be a single nucleotide polymorphism (SNP), such as wherein one base in the subject's genetic sequence has been substituted for another; there may be more extensive substitutions of a plurality of nucleotides; there may be an insertion or a deletion, such as where one or a multiplicity of bases have been added to or deleted from the subject's genetic sequence, and/or there may be a structural variant, e.g., such as caused by the crossing of legs of two chromosomes, and/or there may simply be an offset causing a shift in the sequence. In various instances, a variant call file containing all the variations of the subject's genetic sequence to the reference sequence is generated. More particularly, in various embodiments, the methods of the disclosure include generating a variant call file (VCF) identifying one or more, e.g., all of the genetic variants in the individual who's DNA was sequenced, e.g., relevant to one or more reference genomes. The VCF in its basic form is a list of locations of variants and their type: e.g., chromosome 3, at position X, an "A" is substituted for a "T", etc.

[0186] However, as indicated above, in order to generate such a variant call file, the genome of the subject must be sequenced and rebuilt prior to determining its variants. There are, however, several problems that may occur when attempting to generate such an assembly. As noted above, there may be problems with the chemistry, the sequencing machine, and/or human error that occurs in the sequencing process. Additionally, there may be genetic artifacts that make such reconstructions problematic. For instance, a problem with performing such assemblies is that there are sometimes huge portions of the genome that repeat themselves, such as long sections of the genome that include the same strings of nucleotides. Hence, because any genetic sequence is not unique everywhere, it may be difficult to determine where in the genome an identified read actually maps and aligns.

[0187] For instance, dependent on the sequencing protocol employed shorter or longer reads may be produced. Longer reads are useful in that the longer the read the less likely it is to show up in multiple locations in the genome. Having fewer possible locations to evaluate can also speed up the system. However, the longer the reads the more problematic they may be because the more likely they are to include a real or false variation, e.g., caused by an SNP, InDel (insertion or deletion), or a machine error, or the like, resulting in a no match between the read and the reference genome. On the other hand, shorter reads are useful because the shorter the read the less likely it is to cover a position that codes for a variant. A problem with shorter reads however is that the shorter the read the more likely it is to show up at multiple positions in the genome, thus requiring additional processing time and resources so as to determine which out of all possible positions is the most likely actual position to where it aligns. Ideally what may be achieved, such as by practicing the methods herein disclosed, is that a variant call file may be produced wherein a list of the sequenced genome (the query sequence) is generated that shows where all the variant base pairs are, making sure each variant called is an actual variant and not simply a chemistry or machine read or other human based error.

[0188] There are, therefore, two main possibilities for variation. For one, there is an actual variation at the particular location in question, for instance, where the person's genome is in fact different at a particular location than that of the reference, e.g., there is a natural variation due to an SNP (one base substitution), an Insertion or Deletion (of one or more nucleotides in length), and/or there is a structural variant, such as where the DNA material from one chromosome gets crossed onto a different chromosome or leg, or where a certain region gets copied twice in the DNA. Alternatively, a variation may be caused by there being a problem in the read data, either through chemistry or the machine, sequencer or aligner, or other human error. Accordingly, the methods disclosed herein may be employed in a manner so as to compensate for these types of errors, and more particularly so as to distinguish errors in variation due to chemistry, machine or human, and real variations in the sequenced genome. More specifically, the methods, apparatuses, and systems for employing the same, as here in described, have been developed so as to clearly distinguish between these two different types of variations and therefore to better ensure the accuracy of any call files generated so as to correctly identify true variants.

[0189] Further, in various embodiments, once the subject's genome has been reconstructed and/or a VCF has been generated, such data may then be subjected to tertiary processing so as to interpret it, such as for determining what the data means with respect to identifying what diseases this person may or may have the potential for suffer from and/or for determining what treatments or lifestyle changes this subject may want to employ so as to ameliorate and/or prevent a diseased state. For example, the subject's genetic sequence and/or their variant call file may be analyzed to determine clinically relevant genetic markers that indicate the existence or potential for a diseased state and/or the efficacy of a proposed therapeutic or prophylactic regimen may have on the subject. This data may then be used to provide the subject with one or more therapeutic or prophylactic regimens so as to better the subject's quality of life, such as treating and/or preventing a diseased state.

[0190] More particularly, medical science technologies have advanced in conjunction with the advancement of information technologies, which advancement has enhanced our ability to store and analyze medical data. Hence, once one or more of an individual's genetic variations are determined, such variant call file information can be used to develop medically useful information, which in turn can be used to determine, e.g., using various known statistical analysis models, health related data and/or medical useful information, e.g., for diagnostic purposes, e.g., diagnosing a disease or potential therefore, clinical interpretation (e.g., looking for markers that represent a disease variant), whether the subject should be included or excluded in various clinical trials, and other such purposes. As there are a finite number of diseased states that are caused by genetic malformations, in tertiary processing variants of a certain type, e.g., those known to be related to the onset of diseased states, can be queried for, such as by determining if one or more genetic based diseased markers are included in the variant call file of the subject.

[0191] Consequently, in various instances, the methods herein disclosed may involve analyzing, e.g., scanning, the VCF and/or the generated sequence, against a known disease sequence variant, such as in a data base of genomic markers therefore, so as to identify the presence of the genetic marker in the VCF and/or the generated sequence, and if present to make a call as to the presence or potential for a genetically induced diseased state. As there are a large number of known genetic variations and a large number of individual's suffering from diseases caused by such variations, in some embodiments, the methods disclosed herein may entail the generation of one or more databases linking sequenced data for an entire genome and/or a variant call file pertaining thereto, e.g., such as from an individual or a plurality of individuals, and a diseased state and/or searching the generated databases to determine if a particular subject has a genetic composition that would predispose them to having such diseased state. Such searching may involve a comparison of one entire genome with one or more others, or a fragment of a genome, such as a fragment containing only the variations, to one or more fragments of one or more other genomes such as in a database of reference genomes or fragments thereof.

[0192] Further, it is understood that the genetic sequences to be employed in these manners may be DNA, ssDNA, RNA, mRNA, rRNA, tRNA, or the like. Hence, although throughout the present disclosure various mention is made to various methods and apparatuses for analyzing genomic DNA, in various instances, the systems, apparatuses and methods disclosed herein are equally suitable for performing their respective functions, e.g., analysis, on all types of genetic material including DNA, ssDNA, RNA, mRNA, rRNA, tRNA, and the like. Additionally, in various instances, the methods of the disclosure may include analyzing the generated genetic sequence, e.g., DNA, ssDNA, RNA, mRNA, rRNA, tRNA, and the like, from the subject and determining therefrom the protein variations which are likely to be caused by the genetic sequence and/or determining and/or predicting the potential for a diseased state therefrom, such as due to an error in protein expression. It is to be noted that the genetic sequence obtained can represent an intron or an exon, for instance, the genetic sequence can be for a coding portion of the DNA only, such as where an exome is obtained and using known processing techniques only the coding regions, or non-coding regions, may be sequenced, which can lead to faster sequencing and/or faster processing times, albeit involving a more difficult sample preparation procedure.

[0193] Currently, such steps and analyses herein described are typically performed in various distinct and unrelated steps often employing different analytic machines at different locations. Accordingly, in various aspects the methods and systems of the disclosure are performed by a single apparatus and/or at one location, such as in conjunction with an automated sequencer or other apparatus configured to generate genetic sequence data. In various instances, a plurality of apparatuses may be employed at the same location, or a multiplicity of remote locations, and in some instances, the methods may involve two or more processing units being deployed at two or more locations.

[0194] For instance, in various aspects a pipeline may be provided wherein the pipeline includes performing one or more analytic functions, as described herein, on a genomic genetic sequence of one or more individuals, such as data obtained in a digital, e.g., FASTQ, file format from an automated sequencer. A typical pipeline to be executed may include one or more of sequencing genetic material, such as a portion or an entire genome, of one or more subjects, which genetic material may include DNA, ssDNA, RNA, rRNA, tRNA, and the like, and/or in some instances the genetic material may represent coding or non-coding regions, such as exomes, episomes of the DNA. The pipeline may include one or more of performing a base calling and/or error correction operation, such as on the digitized genetic data, and/or may include one or more of performing a mapping, an alignment, and/or a sorting function on the genetic data. In certain instances, the pipeline may include performing one or more of a realignment, a deduplication, a base quality or score recalibration, a reduction and/or compression, and/or a decompression on the digitized genetic data. In certain instances the pipeline may include performing a variant calling operation on the genetic data.

[0195] Therefore, in various instances, a pipeline of the disclosure may include one or more modules, wherein the modules are configured for performing one or more functions, such as a base calling and/or error correction operation and/or a mapping and/or an alignment and/or a sorting function on genetic data, e.g., sequenced genetic data. And in various instances, the pipeline may include one or more modules, wherein the modules are configured for performing one more of a local realignment, a deduplication, a base quality score recalibration, a variant calling, a reduction and/or compression, and/or a decompression on the genetic data. Many of these modules may either be performed by software or on hardware or remotely, e.g., via software and/or hardware, such as on the cloud or a remote server and/or server bank.

[0196] Additionally, many of these steps and/or modules of the pipeline are optional and/or can be arranged in any logical order and/or omitted entirely. For instance, the software and/or hardware disclosed herein may or may not include a base calling or sequence correction algorithm, such as where there may be concern that such functions may result in a statistical bias. Consequently the system will include or will not include the base calling and/or sequence correction function, respectively, dependent on the level of accuracy and/or efficiency desired. And as indicated above, one or more of the pipeline functions may be employed in the generation of a genomic sequence of a subject such as through a reference based genomic reconstruction. Also as indicated above, in certain instances, the output from the pipeline is a variant call file indicating a portion or all the variants in a genome or a portion thereof.

[0197] Accordingly, as indicated above, the output of performing a sequencing protocol, such as one or more of those set forth above, is typically a digital representation of the subject's genetic material, such as in a FASTQ file format. However, an autorad that has been digitally transcribed may also be employed. More particularly, the output from a sequencing protocol may include a plurality of reads, where each read includes a sequence, e.g., a string, of nucleotides where the position of every nucleotide has been called, and a quality score representing the probability that the called nucleotide is wrong. However, the quality of these outputs may be improved by various pre-processing protocols so as to achieve higher quality of scores, which one or more of such protocols may be employed in the methods disclosed herein.

[0198] For instance, in certain instances, the raw FASTQ file data may be processed to clean up the initial base calls obtained from the sequencer/reader, such as in a primary processing stage, e.g., prior to the secondary processing described herein above. Specifically, the sequencer/reader typically analyzes the sequencing data, such as the fluorescent or electromagnetic data, indicating which nucleotide is at what position, and converts the image data into a base call with a quality score, such as where the quality score is based on the comparative brightness of the fluorescence at each position. A specialized algorithm may be employed, such as in a primary processing stage, to correctly analyze these distinctions in fluorescence and/or electromagnetism, so as to more accurately make the appropriate base call. As indicated above, this step may be included in a pipeline of steps and may be implemented via software or hardware or both, however, in this instance would be part of a primary processing platform.

[0199] An additional preprocessing step may include an error correction function, which may include an attempt to take the millions to billions of reads in the FASTQ file and correct some proportion of any mechanical sequencing error with the information pertaining to the base call and quality score available prior to any further processing such as mapping, alignment, and/or sorting functions, etc. For instance, the reads within the FASTQ file may be analyzed to determine if there are any sub-sequences in any of the reads that appear in other reads, which because of the duplicate coverage can increase confidence that the subsequences in the reads may be correct. This may be implemented by building a hash table containing all possible k-mers of a selected length, k, from every read, and storing with each one its frequency and also which bases immediately follow it and with what probability. Then, using the hash table each read can be rescanned. As each k-mer in a particular read is looked up in the hash table, and evaluation can be made as to whether the base immediately following that k-mer is likely to be correct or not. If it is unlikely, then it can be replaced with the most likely one to follow from the table. Subsequent k-mers for that read will then include the corrected base as the value at that position and the process is repeated. This can be highly effective in correcting errors because oversampling enables gathering accurate statistics for predicting what comes next after each k-mer. However, as indicated above, such corrections could add statistical biasing to the system, such as due to false corrections, to the data, and so these procedures can be skipped if desired.

[0200] Accordingly, in accordance with the aspects of the disclosure, in various instances, the methods, apparatuses, and/or systems of the disclosure, may include obtaining read data, that either have or have not been preprocessed, such as by being obtained directly from a FASTQ file of an automated sequencer, and subjecting the obtained data to one or more of a mapping, aligning, and/or sorting function. The performance of such functions may be useful, for instance, because, as set forth above, in various instances, the sequencing data typically generated by various automated sequencers, e.g., reads, have lengths that are substantially shorter than the entire genomic sequence being analyzed, and since the human genome typically has a multiplicity of repetitive sections, and is known to have various repeating patterns in it, there may be therefore a multiplicity of locations that any given read sequence may correspond to a segment in the human genome. Consequently, given all the possibilities a given read may match to the sequence of the genome, such as because of various repeating sequences in the genome, etc. the raw read data may not clearly indicate which one of the possibilities is in fact the correct location from which it was derived. Hence, for each read it will need to be determined to where in the genome the reads actually map. Additionally, it may also be useful to determine the sequential alignment of the reads, so as to determine the actual sequence identity of the subject, and/or it may also be useful to determine the chromosomal location for each portion of the sequence.

[0201] In various instances, the methods of the disclosure may be directed to mapping, aligning, and/or sorting the raw read data of the FASTQ files so as to find all the likely places that a given read may be aligned, and/or to determine the actual sequence identify of a subject, and/or to determine the chromosome location for each portion of the sequence. For example, mapping may be employed so as to map the generated reads to the reference genome and thereby find the location where each read appears to match well to the genome, e.g., finding all the places where there might be a good score for aligning any given read to the reference genome. Mapping therefore may involve taking one or more, e.g., all, of the raw or preprocessed reads received from the FASTQ file and comparing the reads with one or more reference genomes and determining where the read may match with the reference genome(s). In its basic from, mapping involves finding the location(s) in the reference genome where one or more of the FASTQ reads obtained from the sequencer appears to match.

[0202] Likewise, alignment may be employed so as to evaluate all the candidate locations of the individual reads against a window of the reference genome to determine where and how the read sequences best align to the genome. However, performing an alignment may be difficult due to substitutions, insertions, deletions, structural variations, and the like which may prevent the read from aligning exactly. There are, therefore, several different ways to get an alignment, but to do so may require making changes in the read, where each change that needs to be made to get the appropriate alignment results in a lower confidence score. For instance, any given read may have substitutions, insertions, and/or deletions as compared to the reference genome, and these variations need to be accounted for in performing an alignment.

[0203] Accordingly, along with the predicted alignment a probability score that the predicted alignment is correct may also be given. This score indicates the best alignment for any given read amongst multiple locations where that read may align. For example, the alignment score is predicated upon how well a given read matches a potential map location and may include stretching, condensing, and changing bits and pieces of the read so as to get the best alignment.

[0204] The score will reflect all the ways the read was changed so as to accommodate the reference. For instance, in order to generate an alignment between the read and the reference one or more gaps in the read may need to be inserted, wherein the insertion of each gap represents a deletion in the read over the reference. Likewise, deletions may need to be made in the read, wherein each deletion represents an insertion in the read over the reference. Such iterations may be performed with respect to changing the reference in view of the read, if desired. Additionally, various bases may need to be changed such as due to one or more substitutions. Each of these changes are made to make the read(s) more exactly align to the reference, but each change comes with a cost to the quality score, which score is a measure as to how well the entire read matches to some region of the reference. The confidence in such quality scores is then determined by looking at all the locations the read can be made to map to the genome and comparing the scores at each location, and choosing the one with the highest score. More particularly, where there are multiple positions with high quality scores, then confidence is low, but where the difference between the first and second best scores is large, then confidence is high. At the end, all the proposed reads and confidence scores are evaluated and the best fit is selected.

[0205] Once the reads are assigned a position relative to the reference genome, which consists of identifying to which chromosome the read belongs and its offset from the beginning of that chromosome, they may be sorted, such as by position. This enables downstream, e.g., tertiary, analyses to take advantage of the various oversampling protocols described herein. All of the reads that overlap a given position in the genome may be adjacent to each other after sorting and they can be piled up and readily examined to determine if the majority of them agree with the reference value or not. If they do not, as indicated above, a variant can be flagged.

[0206] As indicated above, the FASTQ file obtained from the sequencer is comprised of a plurality, e.g., millions to a billion or more, of reads consisting of short strings of nucleotide sequence data representing a portion or the entire genome of an individual. Mapping, in general, involves plotting the reads to all the locations in the reference genome to where there is a match. For example, dependent on the size of the read there may be one or a plurality of locations where the read substantially matches a corresponding sequence on the reference genome. Accordingly, the mapping and/or other functions disclosed herein may be configured for determining where out of all the possible locations one or more reads may match to in the reference genome is actually the true location to where they map.

[0207] It is possible to compare every read with every position in the 3.2 billion reference genome to determine where, if any, the reads match to the reference genome. This may be done, for instance, where the read lengths approach about 100,000 nucleotides, about 200,000 nucleotides, about 400,000 nucleotides, about 500,000 nucleotides, even about 1,000,000 or more nucleotides in length. However, where the reads are substantially shorter in length, such as where there are 50 million reads or more, e.g., 1 billion reads, this process could take a very long time and require a large amount of computing resources. Accordingly, there are several methods, such as described herein, that have been developed for aligning the FASTQ reads to the reference genome in a much quicker manner. For instance, as disclosed above, one or more algorithms may be employed so as to map one or more of the reads generated by the sequencer, e.g., as a FASTQ file, and match them to the reference genome, so as to determine where in the reference genome the subject reads potentially map.

[0208] For instance, in various methods, an index of the reference is generated, so that the reads or portions of the reads may be looked up in the index, retrieving indications of locations in the reference, so as to map the reads to the reference. Such an index of the reference can be constructed in various forms and queried in various manners. In some methods, the index may include a prefix and/or a suffix tree. In other various methods, the index may include a Burrows/Wheeler transform of the reference. In further methods, the index may include one or more hash tables, and a hash function may be performed on one or more portions of the reads in an effort to map the reads to the reference. In various instances, one or more of these algorithms may be performed sequentially or at the same time so as to accurately determine where one or more, e.g., a substantial portion or every, read correctly matches with the reference genome.

[0209] Each of these algorithms may have advantages and/or disadvantages. For example, a prefix and/or suffix Tree and/or a Burrows/Wheeler transformation may be performed on the sequence data in such a manner that the index of the reference genome is constructed and/or queried as a tree-like data structure, where starting from a single-base or short subsequence of a read, the subsequence is incrementally extended within the read, each incremental extension stimulating accesses to the index, tracing a path through the tree-like data structure, until the subsequence becomes unique enough, e.g., an optimal length has been attained, and/or a leaf node is reached in the tree-like data structure, the leaf or last-accessed tree node indicating one or more positions in the reference genome from which the read may have originated. These algorithms, therefore, typically do not have a fixed length for the read subsequences that may be mapped by querying the index. A hash function, however, often employs a fixed length comparison unit that may be the entire length of the read, but is often times a length that is some sub-portion thereof, which sub-portion is termed a seed. Such seeds can be shorter or longer, but unlike with the prefix and/or suffix trees and/or the Burrows/Wheeler transformations, the seeds of the reads employed in a hash function are typically of a preselected, fixed length.

[0210] A prefix and/or suffix tree is a data structure that is built up from the reference genome, such that each link from a parent node to a child node is labeled or associated with a nucleotide or sequence of nucleotides, and each path from a root node through various links and nodes traces a path whose associated aggregate nucleotide sequence matches some continuous subsequence of the reference genome. The node reached by such a path is implicitly associated with the reference subsequence traced by its path from the root. Proceeding from the root node, subsequences in a prefix tree grow forward in the reference genome, whereas subsequences in a suffix tree grow backward in the reference genome. Both a prefix tree and a suffix tree may be used in a hybrid prefix/suffix algorithm, so that subsequences may grow in either direction. Prefix and suffix trees may also contain additional links, such as jumping from a node associated with one reference subsequence to another node associated with a shorter reference subsequence.

[0211] For instance, a tree-like data structure serving as an index of the reference genome may be queried by tracing a path through the tree, corresponding to a subsequence of a read being mapped, that is built up by adding nucleotides to the subsequence, using the added nucleotides to select next links to traverse in the tree, and going as deep as necessary until a unique sequence has been generated. This unique sequence may also be termed a seed, and may represent a branch and/or root of the sequence tree data structure. Alternatively, the tree descent may be terminated before the accumulated subsequence is fully unique, so that a seed may map to multiple locations in the reference genome. Particularly, the tree may be built out for every starting position for the reference genome, then the generated reads may be compared against the branches and/or roots of the tree and these sequences may be walked through the tree to find where in the reference genome the read fits. More particularly, the reads of the FASTQ file may be compared to the branches and roots of the reference tree and once matched therewith the location of the reads in the reference genome may be determined. For example, a sample read may be walked along the tree until a position is reached whereby it is determined that the accumulated subsequence is unique enough so as to identify that the read really does align to a particular position in the reference, such as walking through the tree until a leaf node is reached.

[0212] A disadvantage, however, of such a prefix and/or suffix tree is that it is a huge data structure that must be accessed a multiplicity of times as the tree is walked so as to map the reads to the reference genome. An advantage of a hash table function, on the other hand, as described in greater detail herein below, is that once built, it typically only takes one look up to determine where, if anywhere, there may be a match between a seed and the reference. A prefix and/or suffix tree will typically take a plurality of look ups, e.g., 5, 10, 15, 20, 25, 50, 100, 1,000, or more, etc., in determining if and where there is a match. Further, due to the double helix structure of DNA, a reverse complement tree may also need to be built and searched, as the reverse complement to the reference genome may also need to be found. With respect to the above, the data tree is described as being built from the reference genome which is then compared with the reads from the subject's sequenced DNA, however, it is to be understood that the data tree may initially be built from either the reference sequence or the sample reads, or both, and compared one to the other as described above.

[0213] Alternatively, or in addition to employing a prefix or a suffix tree, a Burrows/Wheeler transform can be performed on the data. For instance, a Burrows/Wheeler transform may be used to store a tree-like data structure abstractly equivalent to a prefix and/or suffix tree, in a compact format, such as in the space allocated for storing the reference genome. In various instances, the data stored is not in a tree-like structure, but rather the reference sequence data is in a linear list that may have been scrambled into a different order so as to transform it in a very particular way such that the accompanying algorithm allows the reference to be searched with reference to the sample reads so as to effectively walk the "tree". An advantage of the Burrows/Wheeler transform, such as over a prefix and/or suffix tree, is that it typically requires less memory to store, and an advantage over a hash function is that it supports a variable seed length, and hence it can be searched until a unique sequence is determined and a match found. For instance, as with the prefix/suffix tree, however many nucleotides it takes for a given sequence to be unique, or to map to a sufficiently small number of reference positions, determines the length of the seed. Whereas for a hash table, the seeds are all of the same predetermined length. A disadvantage, however, for the Burrows/Wheeler transform is that it typically requires a multiplicity of lookups, such as two or more look ups, such as for every step down the tree.

[0214] Alternatively, or in addition to utilizing one or both a prefix/suffix tree and/or a Burrows/Wheeler transform on the reference genome and subject sequence data, so as to find where the one maps against the other, another such method involves the production of a hash table index and/or the performance of a hash function. The hash table index may be a large reference structure that is built up from sequences of the reference genome that may then be compared to one or more portions of the read to determine where the one may match to the other. Likewise, the hash table index may be built up from portions of the read that may then be compared to one or more sequences of the reference genome and thereby used to determine where the one may match to the other.

[0215] More particularly, in any of the mapping algorithms described herein, such as for implementation in any of the method steps herein disclosed, one or all three mapping algorithms, or others known in the art, may be employed, in software or hardware, so as to map one or more sequences of a sample of sequenced DNA with one or more sequences of one or more reference genomes. As described herein in greater detail below, all of these operations may be performed via software or by being hardwired, such as into an integrated circuit, such as on a chip, for instance as part of a circuit board. For instance, the functioning of one or more of these algorithms may be embedded onto a chip, such as into a FPGA (field programmable gate array) ASIC (application specific integrated circuit) chip, or Structured ASIC (application specific integrated circuit) chip, and may be optimized so as to perform more efficiently because of their implementation in such hardware.

[0216] Additionally, one or more, e.g., two or all three, of these mapping functions may form a module, such as a mapping module, that may form part of a system, e.g., a pipeline, that is used in a process for determining an actual entire genomic sequence, or a portion thereof, of an individual. The output returned from the performance of a mapping function may be a list of possibilities as to where one or more, e.g., each, read maps to one or more reference genomes. For instance, the output for each mapped read may be a list of possible locations the read may be mapped to a matching sequence in the reference genome. In various embodiments, an exact match to the reference for at least a piece, e.g., a seed of the read, if not all of the read may be sought. Accordingly, in various instances, it is not necessary for all portions of all the reads to match exactly to all the portions of the reference genome.

[0217] Further, one or all of these functions may be programmed in such a manner that exact or approximate matching and/or editing, such as editing of the results, may be performed. Hence, all of these processes can be configured to do inexact matching as well, where desired, such as in accordance with a preselected variance, such as 80% matching, 85% matching, 90% matching, 95% matching, 99% matching, or more. However, as described in greater detail herein below, inexact matching may be a lot more expensive such as in time and processing power requirements, because it may require any number of edits, e.g., where the edit may be a SNP or insertion or deletion of one or more bases, e.g., 1 or 2 or 3 or 5 or more edits, to be performed so as to achieve an acceptable match. Such editing is likely to be used more extensively in implementing hashing protocols or when implementing prefix and/or suffix trees and/or performing a Burrows/Wheeler transform.

[0218] With respect to hash tables, a hash table may be produced in many different ways. In one instance, a hash table may be built by breaking the reference genome into segments of standard length, e.g., seeds of about 16 to about 30 nucleotides or more in length, such as about 18 to about 28 nucleotides, formatting them into a searchable table, and making an index of all the reference segments from which sequenced DNA, e.g., one or more reads, or a portion thereof, may be compared to determine matching. More particularly, a hash table index may be generated by breaking down the reference genome into segments of nucleotide sequences of known, uniform length, e.g., seeds, and storing them in random order into individual cubicles in the reference table. This may be done for a portion or the entire reference genome so as to build an actual reference index table that may be used to compare portions of the reference genome with portions of one or more reads, such as from a FASTQ file, for the purpose of determining matching.

[0219] This method may then be repeated in approximately the same manner for a portion, e.g., a majority or all, of the reads in the FASTQ file, so as to generate seeds of the appropriate, e.g., selected, length. For instance, the reads of the FASTQ file may be used to produce seeds of a predetermined length, which seeds may be converted into binary form and fed through a hash function and fit into a hash table index where the binary form of the seeds may match up with the binary segments of the reference genome, so as to give the location as to where in the genome the sample seeds match with the position in the reference genome.

[0220] For example, where the read is approximately 100 bases long, a typical seed may be about half or a about a third, e.g., about 27 to about 30 bases, as long. Hence, in such an instance, for each read a multiplicity of seeds, e.g., approximately 3 or 4 seeds dependent on the length of the read and/or the length of the seeds, may be generated to cover the read. Each seed may then be converted into a binary form and/or then be fed into the hash table and a possible result as to its position with respect to the reference genome may be obtained. In such instances, the entire read need not be compared to every possible position in the entire reference genome, rather only a portion of the reads, e.g., one or more of the generated sample seeds per read, need only be compared such as to an index containing equivalent seed portions of the reference genome. Hence, in various instances, a hash table may be configured such that by only one memory look up it can typically be determined where the sample seed and therefore read is positioned relative to the reference genome. However, in certain instances, it may be desirable to perform a hash function and look up on one or more overlapping sections of seeds from one read. In such instances, the seeds to be generated may be formed in such a manner that at least a portion of their sequence overlaps one another. This may be useful for instance in getting around machine and/or human errors or differences between the subject and the reference genome and may promote exact matching.

[0221] In certain instances, the building of the hash table as well as the performance of one or more of the various comparisons is executed by the hash function. The hash function is in part a scrambler. It takes an input and gives what appears to be a random order to it. In this instance, the hash function scrambler breaks down the reference genome into segments of a preselected length and places them randomly in the hash table. The data may then be stored evenly across the whole storage space. Alternatively, the storage space may be segmented and/or storage therein may be weighted differently. More particularly, the hash function is a function that takes any input and gives a number, such as a binary pattern out, which number may typically random except that for any one given input the same output is always returned. Hence, even if two inputs that are fed into the hash table are almost the same, because they are not an exact match, two completely, randomly different outputs will be returned.

[0222] Further, since genetic material may be composed of four basic nucleotides, e.g., “A”, “C”, “G”, and “T” (or “U” in the case of RNA), the individual nucleotides of the sequences, e.g., the reference segments and or reads, or portions thereof, to be fed into the hash table may be digitized and represented in binary format, such as where each of the four bases represents a two bit digital code, e.g., “A”=00, “C”=01, “G”=11, and “T”/“U”=10. In certain instances, it is this binary “seed” value that is then randomly placed in the hash table at a known location having a value equal to its binary representation. The hash function, therefore, works to break down the reference genome into binary representations of reference seeds and inserts each binary seed data into a random space, e.g., cubicle, in the hash table based on its numeric value. Along with this digital binary code, e.g., access key, each cubicle may also include the actual entry points to where the segment originated from in the actual reference genome, e.g., the reference position. The reference position therefore may be a number indicating the position of the original reference seed in the genome. This may also be done for overlapping positions, which are put into the table in random order but at known location, such as by the hash function. In a manner such as this, a hash table index may be generated, wherein the index includes the digital binary code for a portion or all of a plurality of segments of one or more reference genomes, which may then be referenced by one or more sequences of genetic material, e.g., one or more reads, or portions thereof, from one or more individuals.

[0223] When implementing the hash table and/or function as a module, such as a module in a pipeline of modules, on software (such as where the bit width is 2× the number of bases in the seed described above) and/or hardware, as referenced above, the hash table can be built so that the binary representation of the reference seeds can be any bit width desired. As the seeds can be long or short, the binary representations can be greater or lesser, but typically the seed length should be chosen so as to be long enough to be unique, but not too long that it is too hard to find matches between the seeds of the genome reference and the seeds of the sample reads, such as because of errors or variants. For instance, as indicated above, the human genome is made up of about 3.1 billion base pairs, and a typical read may be about 100 nucleotides in length. Hence, a useful seed length may be between about 16 or about 18 nucleotides or less in length to about 28 or about 30 nucleotides or more in length. For example, in certain instances, the seed length may be a segment of 20 nucleotides in length. In other instances, the seed length may be a segment of 28 nucleotides in length.

[0224] Consequently, where the seed length is a segment of 20 nucleotides, each segment may be represented digitally by a 40 bit output, e.g., a 40 bit binary representation of the seed. For example, where 2 bits are selected to represent each nucleotide, e.g., such as where A=00, C=01, G=10, and T=11, a seed of 20 nucleotides×2 bits per nucleotide=a 40 bit (5 byte) vector, e.g., number. Where the seed length may be 28 nucleotides in length, the digital, e.g., binary, representation of the seed may be a 56 bit vector. Hence, where the seed length is approximately 28 nucleotides in length, 56 bits can be employed to handle a 28 nucleotide seed length. More particularly, where the 56 bits represents the binary form of the seeds of the reference genome that have been randomly positioned in the hash table, a further 56 bits can be used to digitally represent the seeds of the read that are to be matched against the seeds of the reference. These 56 bits may be run through a polynomial that converts the 56 bits in to 56 bits out in a 1:1 correspondence. Without increasing or decreasing the number of bits of output, performing this operation randomizes the storage location of adjacent input values so that the various seed values will be uniformly distributed among all possible storage locations. This also serves to minimize collisions among values that hash to the same location. In particular, in a typical hash table implementation described herein, only a portion of the 56 bits is used as a lookup address to select a storage location and the remaining bits are stored in that location for confirmation of a match. If a hashing function were not used, a great many patterns having the same address bits, but different stored bits would have to share the same hash location.

[0225] More specifically, there is similarity between the way the hash table is constructed, e.g., by software and/or hardware placing the reference genome seeds randomly in the hash table, and the way the hash table is accessed by the seeds of the reads being hashed such that they both access the table in the same way. Hence, seeds of the reference and seeds of the sample read that are the same, e.g., have the same binary code, will end up in the same location, e.g., address, in the table because they access the hash table in the same manner, e.g., for the same input pattern. This is the fastest known method for performing a pattern match. Each lookup takes a nearly constant amount of time to perform. This may be contrasted with a Burrows-Wheeler method which may require many probes (the number may vary depending on how many bits are required to find a unique pattern) per query to find a match, or a binary search method that takes $\log_2(N)$ probes where N is the number of seed patterns in the table.

[0226] Further, even though the hash function can break the reference genome down into segments of seeds of any given length, e.g., 28 base pairs, and can then convert the seeds into a digital, e.g., binary, representation of 56 bits, not all 56 bits need be accessed entirely at the same time or in the same way. For instance, the hash function can be implemented in such a manner that the address for each seed is designated by a number less than 56 bits, such as about 20 to about 45 bits, such as about 25 to about 40 bits, such as about 28 to about 35 bits, including about 28 to about 30 bits may be used as an initial key or address so as to access the hash table.

[0227] For example, in certain instances, about 26 to about 29 bits may be used as a primary access key for the hash table, leaving about 27 to about 30 bits left over, which may be employed as a means for double checking the first key, e.g., if both the first and second keys arrive at the same cell in the hash table, then it is relatively clear that said location is where they belong. Specifically, in order to save space and reduce the memory requirements and/or processing time of the hash module, such as when the hash table and/or hash function are implemented in hardware, the about 26 to about 29 bits representing the primary access key derived from the original 56 bits representing the digitized seed of a particular sequenced read may be employed by the hashing function to comprise the primary address, leaving about 27 to about 30 bits that can be used in a double checking method.

[0228] More particularly, in various instances, about 26 to about 29 bits from the 56 bits representing the binary form of a reference seed may be employed to comprise a primary address, which designated 26 to 29 bits may then be given a randomized location in the hash table, which in turn may then be populated with the location of where the reference seed originally came from along with the remaining 27 to 30 bits of the seed so that an exact match may be ascertained. The query seeds representing the reads of the subject genome converted into binary form may also be hashed by the same function in such a manner that they as well are represented by 29 bits comprising a primary access key. If the 29 bits representing the reference seed are an exact match to the 29 bits representing the query seeds, they both will be directed to the same position in the hash table. If there was an exact match to the reference seed, then we expect to find an entry at that location containing the same remaining 27 to 30 bits. In such an instance, the 29 designated address bits of the reference sequence may then be looked up to identify the position in the reference to where the query read from which the query seed was derived, aligns.

[0229] However, with respect to the left over 27 to 30 bits, these bits may represent a secondary access key that may also be imported into the hash table as well, such as for the purpose of ensuring the results of the first 26 to 29 bits of the primary access key. Because the hash table represents a perfect 1:1 scrambling of the 28 nucleotide/56 bit sequence, and only about 26 to about 29 of the bits are used to determine the address, these 26 to 29 bits of the primary access key have basically been checked, thereby determining the correct address in a first go around. This data, therefore, does not need to be confirmed. However, the remaining about 27 to about 30 bits of the secondary access key must be checked. Accordingly, the remaining about 27 to 30 bits of the query seeds are inserted into the hash table as a means for completing the match. Such an implementation may be shorter than storing the 56 bit whole key, and thus, saves space and reduces over all memory requirements and processing time of the module.

[0230] The hash table, therefore, can be configured as an index where known sequences of one or more reference genomes that have been broken down into sequences of predetermined lengths, e.g., seeds, such as of 28 nucleotides in length, are organized into a table randomly, and one or more sequenced reads, or "seed" portions thereof, derived from the sequencing of a subject's genomic DNA or RNA, may be passed through the hash table index, such as in accordance with a hash function, so as to look up the seed in the index, and one or more positions, e.g., locations in the reference genome, may be obtained from the table where the sample seed matches positions in the reference genome. Using a brute force linear search to scan the reference genome for locations where a seed matches, over 3 billion locations would have to be checked. However, by using a hashing approach, each seed lookup can occur in approximately a constant amount of time. Often, the location can be ascertained in a single access. In cases where multiple seeds map to the same location in the table, a few additional accesses may be made to find the seed being currently looked up. Hence, even though there can be 30M or more possible locations for a given 100 nucleotide length read to match up to, with respect to a reference genome, the hash table and hash function can quickly determine where that read is going to show up in the reference genome. By using a hash table index, therefore, it is not necessary to search the whole reference genome to determine where the read aligns.

[0231] As indicted above, chromosomes have a double helix structure that is comprised of two opposed, complementary strands of nucleic acid sequences that are bound together so as to form the double helix. For instance, when the double helix structure is formed these complementary base pairs bind one with the other in accordance with the following formula: "A" binds to "T", and "G" binds to "C". Accordingly, this results in two equal and opposite strands of nucleic acid sequences that are the complement of each other. More particularly, the bases of a nucleotide sequence of one strand will be mirrored by their complementary bases on the opposed strand resulting in two complementary strands. However, transcription of DNA takes place in one direction only, starting from one end of the DNA and moving towards the other. Hence, as it turns out, for one strand of the DNA, transcription takes place in one direction, and for its complement strand, transcription takes place in the opposite direction. Consequently, the two strands of DNA sequences turn out to be reverse complemented, that is if the sequence order of one strand of the DNA is compared to the other what can be seen is two strands where the nucleotide letters of one strand are switched for their complement in the other strand, e.g., "As" for "Ts" and "Gs" for "Cs" and vice versa, and their order is reversed.

[0232] Because of the double helix structure of the DNA, during the sample prep step prior to sequencing the DNA, the chromosomes are pulled apart, e.g., de natured, separated into separate strands, and then lysed into smaller segments of a predetermined length, e.g., of 100-300 bases long, which are then sequenced. It is possible to separate the strands prior to sequencing so that only one strand is sequenced, but typically the strands of DNA are not separated and so both strands of DNA are sequenced. Accordingly, in such an instance, about half of the reads in the FASTQ file may be reverse complemented.

[0233] Of course, both strands of the reference genome, e.g., the complement and the reverse complement, may be processed and hashed as described above, however this would make the hash table twice as big, and make the performance of the hash function take twice as long, e.g., it could require about twice the amount of processing to compare both complement and reverse complemented sequences of the two genomic sequences. Accordingly, to save memory space, reduce processing power, and/or decrease the time of processing, in various instances, only one strand of the model genomic DNA need be stored in the hash table as a reference.

[0234] However, because in accordance with typical sequencing protocols, such as where the two strands of the subject DNA have not been isolated from one another, any read generated from the sequenced DNA can be from either strand, the complement or its reverse complement, it may be difficult to determine which strand is being processed, the complement of the reverse complement. More specifically, in various instances, since only one strand of the reference genome need be used to generate the hash table, half of the reads generated by the sequencing protocol may not match the particular strand, e.g., either the complement or its reverse complement, of the model genome reference, e.g., because half the time the read being processed is a reverse complement with respect to the hashed segments of the reference genome. Hence, only the reads generated from one strand of the DNA will match the indexed sequences of the reference genome, while the reads generated from the other strand will theoretically be their reverse complements and will not match anywhere in the reference genome. Further, an additional complication can be that for any given read that is reverse complemented to the stored reference genome strand, the read may still, erroneously, match to a portion of the reference genome, such as by mere chance. In view of the above, in order for mapping to proceed efficiently, in various instances, it not only must be determined where the read matches in the reference genome it must also be determined if the read is reverse complemented. Therefore, the hash table and/or function module should be constructed so as to be able to minimize these complications and/or the types of errors that may result therefrom.

[0235] For instance, as indicated above, in one instance, the hash table could be populated with both the complement and the reverse complement for the reference genome so that every read or its reverse complement of the subject's sequenced DNA can be matched to its respective strand in the genomic reference DNA. In such an instance, for any given seed in a read, the seed should theoretically match with one strand or the other, the complement or the reverse complement of the reference, assuming no errors or variations. However, storing both strands of the reference genome in the hash index can require about twice as much storage space (e.g., instead of 32 gigabytes 64 gigabytes may be necessary), and may require twice the amount of processing resources and/or twice as much time for processing. Further, such a solution doesn't solve the problem of palindromes that can match in both directions, e.g., the complement and reverse complement strands.

[0236] Accordingly, although the hash table index may be constructed to include both strands of the genomic reference sequence. In various instances, the hash table may be constructed so as to only include one strand of the model genome as a reference. This may be useful because storing the hash table in memory will require half of the storage and/or processing resources than would be required if both strands were to be stored and processed, and thus, the time required for a look up should also require less time. However, storing only one strand of the genome as a reference could cause complications because, as indicated above, where the sequenced subject DNA is double stranded, it is not typically known from which strand any given read was generated. In such an instance, therefore, the hash table should be constructed to account for the fact the read being mapped may be from either strand and thus can be the complement or reverse complement of the stored segments of the reference genome.

[0237] Accordingly, in various instances, such as where only one orientation of seeds from the reference are populated into the hash table, when performing the hash function on the seeds generated from the reads of the FASTQ file, the seed may first be looked up in its present orientation, and/or may then be reverse complemented and the reverse complement may be looked up. This may require two looks up in the hash index, e.g., twice as many, but one of the seed or its reverse complement should match its complementary segment in the reference genome, assuming no errors or variations, and it should reduce the overall processing resources, e.g., less memory is used, as well as reducing time, e.g., not as many sequences need to be compared.

[0238] More particularly, such as where a seed in one particular orientation is comprised of 28 nucleotides, e.g., digitally represented in a 56 bit binary format, as described above, the seed can be reverse complemented and the reverse complement can also be represented digitally in a 56 bit binary format. The binary format for each representation of the seed sequence and its complement results in a number, e.g., an integer, having a value represented by that number. These two values, e.g., the two integers, may be compared and the number with the higher or lower value, e.g., higher or lower absolute value, may be selected as the canonical choice of orientation and that is the one that can be stored in the hash table and/or subjected to the hash function. For instance, in certain instances, the number with the higher value may be selected for being processed by the hash function.

[0239] Another method that may be employed is to construct seeds wherein each seed is comprised of an odd number of bases. The canonical orientation to be selected then may be those strands having a middle base being an "A" or a "G", but not a "T" or a "C", or vice versa. The hash function then will be performed on the seeds meeting the requirements of the canonical orientation. In such a manner, it is only the two bits representing the middle base that needs to be compared to see which has the higher value and it is only the 2 bits of that sequence that are looked up. Hence, you only have to look at the bits representing the middle two bases. Typically, this can work well because if the seed is an odd length, then it always reverse complements the center base. However, although this may work for odd seed lengths, hashing those seeds having a higher, or lower, value, as described above, should work for all seed lengths, albeit such a method may require having to process, e.g., look up, more bits of data.

[0240] These methods may be performed for any number of seeds, e.g., all seeds of the reference and/or any number of seeds, e.g., all, derived from all or a portion of the reads of the FASTQ file. Approximately half of the time the binary representation of the seeds of a given orientation, e.g., the complement, will have a higher value, and approximately half the time the binary representation of the seeds of the opposite orientation, e.g., the reverse complement, will have the higher value. But, when looking at the binary numbers, whichever one has the higher value, that is the one that gets fed into the hash table. For instance, the binary integers for each read and its complement may be compared, and the sequence having the first 1 encountered is the one of the two strands selected to be stored as the strand in the hash table and/or be subjected to the hash function. If both strands have a first 1 in the same position, then the strand having the second 1 that comes first is selected, and so on. Of course, the read with the lower value may also be selected, in which case the strand having the first and/or larger number of initial 0's will be selected. An indication, e.g., a flag, may also be inserted into the hash table where the flag indicates which orientation, complement or reverse complement, the stored and/or hashed strand represents, e.g., a 1RC flag, if reverse complemented.

[0241] More particularly, when performing the hash function and accessing the hash table, seeds from the genomic reference DNA and seeds derived from the reads of the sequence data are subjected to these same operations, such as converted into binary form and compared with its reverse complement where the integers having the higher, or lower, values are selected as the canonical orientations and subjected to the hash function and fed into the hash table to be looked up and matched against each other. However, because it is the same operation being performed in substantially the same manner on the reference sequences and the read sequences, the same record will be derived, if the two sequences, the reference and the subject seeds, have the same sequence to begin with, even if one was reverse complemented, they will all be directed to the same cell in the hash table.

[0242] Consequently, if a certain seed in the reference having a given sequence in a particular orientation is converted to binary form and hashed, and then a seed derived from a sample read having the same sequence, but in its reverse orientation, e.g., reverse complemented, and it is subjected to the above protocols, because of the above disclosed methods, when the binary value is determined and the hash function performed, the look up will be directed to the very same address in the hash table as if the hash function were performed on the complimentary seed to begin with. Hence, in this manner it doesn't matter which orientation the seed being processed is in because it will always be directed to the same address.

[0243] Therefore, in a manner such as this, the methods herein disclosed are able to hash and thereby determine the location of the seed within the table despite its orientation, and because of the flag in the record it will also be known if any given seeds is reverse complemented. For instance, it will be known if the seed was flipped from the reference and it will also be known if the seed derived from the subject read had to be flipped as well. Consequently, if the decision was the same on both sides then the orientation is the same between the read and the reference. However, if one side is flipped and the other is not, then it can be concluded that the read maps reverse complemented to the reference. Hence, by using a hash table it may be determined where in the genome a given read, or portion thereof, e.g., a seed, matches and/or if it is reverse complimented. Further, it is to be understood that although the above is described with respect to generating the hash table from the reference genome and performing various ancillary hash function processes on the seeds generated from the reads, e.g., from a FASTQ file, the system can also be structured such that the hash table index is generated from seeds derived from the reads of the subject's sequenced DNA, and the various ancillary hash function processes, as herein described, are performed on seeds generated from the reference genome.

[0244] As set forth above, an advantage of employing a hash table and/or a hash function is that by employing the use of seeds, a majority of the reads of the sequenced DNA can be matched to the reference genome often by employing single hash lookups, and in various instances, not all seeds derived from a read need be hashed and/or looked up. Seeds may be of any suitable length, such as relatively short, e.g., 16 nucleotides or less, such as about 20 nucleotides, such as about 24 nucleotides, such as about 28 nucleotides, such as about 30 or about 40 or about 50, or 75 or about 100 nucleotides, or even up to 250 or 500, or 750, or even 999 or even about 1,000 nucleotides in length; or relatively long such as over about 1,000 nucleotides or over about 10,000, or over about 100,000 or over 1,000,000 or more nucleotides in length. However, as described above, there are some disadvantages to using seeds, such as in a hash table, in particular with respect to selecting seeds of the appropriate length.

[0245] For instance, any suitable seed length may be employed in a mapping function, however there are advantages and disadvantages of using relatively short or relatively long seed lengths. For example, the shorter the seed length the less likely it is to incorporate an error or a variation that can prevent finding a match within the hash table. However, the shorter the seed length, the less unique it is, and the more matching is to be expected between the seeds of the reference genome and the seeds derived from the reads of the subject's sequenced DNA. Further, the shorter the seed length the more lookups will have to be performed by the hash function, taking more time and increased processing power.

[0246] On the other hand, the longer the seed length the more unique it is and the less likely there is to be multiple matching positions between the seeds between the seeds of the reference and the query. Also, with a longer seed, there need be fewer seeds within the read, so fewer look ups, thereby taking less time and requiring less processing power. The longer the seed, however, the more likely it is that the seeds derived from the sequenced DNA may include an error, such as a sequencing error and/or may incorporate a variation as compared to the reference thus preventing a match from being made. Longer seeds further have the disadvantage of being more likely to hit the end of the read and/or the end of the chromosome. Hence, where a seed is only 20-100 nucleotides in length, there may be several matches within the hash table, however, where the seed is 1,000 or more nucleotides in length there may be much fewer matches, but there may be no matches at all.

[0247] There are some methods for helping to minimize these issues. One method is to ensure there is appropriate oversampling generated in the DNA processing steps prior to sequencing. For instance, as it is known that there is typically at least one variation within every 1,000 base pairs, the seed length may be chosen to maximize matches, while at the same time minimizing non-matches due to the incorporation of errors and/or variants. Additionally, the use of oversampling, such as in the pre-sequencing and/or sequencing steps, can be employed as a further method for minimizing various problems that are inherent to using seeds, such as within a hash function.

[0248] As indicated above, oversampling produces pileups. Pileups are those collections of reads that map in an overlapping fashion generally to the same place in the genome. For the majority of sample reads, such pileups may not be necessary, such as where the reads, and/or seeds generated therefrom, do not include a variant and/or do not map to multiple positions in the hash table (e.g., are not exactly duplicated in the genome). However, for those reads and/or seeds that may include a variant and/or an error and/or other mismatch between the seed and/or read and the reference genome, the production of pileups for any given region of the genome may be useful. For instance, even though only one exact hit between a seed generated from a read of the sample genome is necessary so as to be able to map the sample read to the reference genome, however, the fact that there may be a machine error or a true variant in the sample DNA sequence that could prevent such an exact match between the read and the reference from occurring, often times makes the production of overlapping pileups in the pre-sequencing and sequencing steps useful.

[0249] For example, for those instances where a sample seed does in fact contain a variant or an error, the production of read pileups may be useful in distinguishing between actual variance and machine and/or chemistry errors. In such an instance, a pileup can be employed to determine whether an apparent variation is in fact a real variation. For instance, if 95% of the reads in the pileup indicate that there is a "C" in a certain position, then odds are that is the correct call, even if the reference genome has a "T" at that location. In such an instance, the mismatch may be due to a SNP, e.g., a substitution of a "C" for a "T" in that position in the genome, where the genetic code for the individual actually varies from that of the reference. In such an instance, the depth of the pileup may be employed so as to compare the overlapping portions of the reads of the pileup at a position where there is variance, and based on the percentage of reads in the pileup having the variance, it can be determined whether the variance is in fact due to an actual variation in the sample sequence. Accordingly, the actual sequence of the reads that best fits the genomic sequence, may in part be determined based on what is reflected in the pileup depths. The disadvantage of using pileups, however, is that it requires more processing time to process all the excess reads and/or seeds generated thereby.

[0250] Another method for minimizing the issues inherent in short or long reads is to employ a secondary hash table along with or in conjunction with the first, e.g., primary hash table. For instance, a second hash table and/or hash function may be employed for those seeds that do not have any hits in the primary hash table, or for those seeds that have multiple hits in the primary hash table. For example, when comparing one seed with another there are several outcomes that may result. In one instance, a no hit, e.g., a no match anywhere between the two sequences, may result, in which case this suggests a possible error or variation such as in the seed of a read of the subject as compared against a seed derived from the reference genome. Or there may be one or a plurality of matches found. If a large number of matches are found, however, this could be problematic.

[0251] For instance, with respect to the primary hash table, if each seed in the reference being hashed appears only a few times, e.g., once, twice, or three times, etc. then there may not be a need for a secondary hash table and/or hash function. However, if one or more of the seeds occurs a greater number of times, e.g., 5, 10, 15, 20, 25, 50, 100, 1,000, or more times, this could be problematic. For example, there are known regions in the sequence of the human genome that have been determined to be mathematically significant in that they are repeated a multiplicity of times. Consequently, any seed mapping to one of these positions, may in fact inadvertently map to a multiplicity of these positions, such as where the seed comprises the nucleotides of the overlapping sequences. In such an instance, determining which out of all the possibilities the seed actually aligns to may be difficult. However, as these repeating regions are known, and/or become known, any seed that would typically map to one or more of these regions may be demarcated to be allocated to a secondary hash table for processing by the first or a secondary hash function, so as to not waste time and processing power trying to use a primary hashing function to determine something that is likely to be indeterminable.

[0252] More particularly, when comparing the seeds of the genomic reference to the seeds generated from the subject's genomic reads, anywhere from 1 to hundreds or even thousands of match positions may result. The present system, however, may be configured to handle a certain number of duplicative matches, such as without the need for further processing steps, such as where the number of matches is below about 50, or below about 40, or below about 30, such as below about 25 or about 20, such as below about 16 matches or below about 10 or about 5 matches. However, if there are more matches of viable hits than this that are returned, then the system can be configured to implement a secondary hash function, e.g., using a secondary hash table.

[0253] Accordingly, rather than placing such seeds known to have an increased likelihood of redundancy in the primary hash table, such seeds can be placed in a secondary hash table, or a secondary region in the first hash table. Additionally, in some instances, a record that doesn't communicate anything about the multiplicity of potential map positions for that seed, but rather communicates a command to access a secondary hash table, e.g., an extend record, can be placed in the primary hash table. For example, the extend record can be an instruction, such as an instruction to extend the primary, e.g. non unique or duplicative, seed length to a longer, more unique seed length, such as by adding on one or more additional bases next to it, e.g., on the end(s) of the seed, to make it a longer seed sequence that can then get hashed and looked up, such as in the secondary table.

[0254] The record can be configured such that it informs or otherwise instructs how much to extend the known redundant seed by a given amount, and may also instruct as to where and/or how to extend the seed. For instance, because the hash table is usually precomputed, e.g., originally constructed from the seeds generated from the reference genome(s), it may be known prior to constructing the table, which, if any, of the seeds generated from the reference genome are going to occur a multiplicity of times. Hence, in various instances, it may be predetermined which seeds are going to need to be shifted over to the secondary hash table. For example, when constructing the hash table index, the characteristics of the reference seed sequences being input into the hash table as an index are known, so for every potential seed it may be determined whether it's a case that is going to give a multiplicity of hits, e.g., from 10-10,000 hits.

[0255] More particularly, in various instances, an algorithm can be performed to determine all the predicted matches a given seed derived from the reference and/or the subject's reads may have. If it is determined that for any particular seed that it is likely to return a multiplicity of matches, a flag, e.g., a record, may be generated, such as within a cell of the hash table, indicating that this particular seed is a high frequency hit. In such an instance, the record can further instruct that the primary hashing of this seed, and such seeds like it, should be skipped over because it is not practical to perform the number, e.g., 20-10,000 or more evaluations on such a seed needed to accurately determine where the seed actually maps. In such an instance, the primary hash function may not be able to accurately determine which position out of all the possible positions to where the seed may match, is the one to where the read actually aligns, and thus for practical purposes, because the seed cannot accurately be mapped at this stage, the primary hash function may not be likely to return a useable result, such as a result indicating accurately where the seed actually matches in the genome.

[0256] In such an instance, the hash function algorithm may be configured to calculate what would need to be done to make the redundant seed more unique. For example, the secondary hash function may determine by how many bases the seed needs to be extended, and in what order, and in what location, so as to ensure that the seed is no longer redundant, but rather suitably unique so as to be hashed. Accordingly, the record may also include an instruction to extend the redundant seed, e.g., extend by two, by four, by six, etc., on one or both ends of the seed so as to achieve a predetermined level of uniqueness. In such a manner as this, seeds that at first appear to be identical can be determined to be non-identical.

[0257] For example, in some instances, a typical record can instruct that the duplicative seed be extended by up to X number of odd or even bases, but in some instances, extended by an even number of bases, such as from about 2 to 4 to about 8 to 16 to about 32 or about 64 or more bases, such as equally on each side. For instance, where the extension is to be by 64 bases, the record could instruct that 32 bases be added on each side of the seed. The number of bases by which the seed is to be extended is configurable and may be any suitable number dependent on how the system is constructed. In certain instances, the secondary hash function may be employed to determine by how many bases the seed should be extended so as to get a more reasonable number of match results back. Therefore, the extension may be to the point of relative uniqueness, such as to where there is only 1, 2, 3, or even up to 16 or 25 or 50 match positions where the pattern shows up. In various instances, extending the seed equally from both ends may be useful such as to avoid problems with reverse reads, but in various instances the seed may be extended by the addition of one or more bases unequally to both sides.

[0258] More particularly, such as in one example, if the seed includes 28 bases, and an extend record, such as an extend record positioned within a cell in the primary hash table, instructs the hash function to extend the seed, such as by 64 bases, then the record may further direct the hash function as to how to extend the seed, such as by adding 32 bases on each side of the seed. However, the extension can take place at any suitable position on the read and may be done in a symmetrical or asymmetrical fashion. In certain instances, the record may instruct the hash function to extend the seed symmetrically because in certain instances such a symmetrical extension may work better, such as with reverse complements, discussed herein. In such an instance, the same number of bases will be added such as to the opposite sides of the seed when extending. Although in other instances extension may be performed by adding an even or an odd number of bases in a non-symmetrical format, and hence, it is not necessary to extend the seed by same number of bases on each side. Typically, the primary hash table is configured such that it is not completely full. For example it is desirable to configure it not to exceed 80% or 90% of its capacity. This is to maintain high performance of the lookup rate. When there are a high number of collisions in hashing seeds to the same location when constructing the table, the storing mechanism will create a chain of references to other locations so that the lookup mechanism will be able to find the one assigned to the overflowed seed. The denser the table, the higher the number of collisions and the longer the chains to be followed to find the actual match.

[0259] In various instances, such as where the initial, redundant seed is 28 bases long, and the record instructs for it to be extended, such as from 18 to 32 to 64 bases, such as on each opposed side of the seed, the digital representation of the seed may be about 64 bases×2 bits per base=128 bits. Accordingly, dependent on how the mapping module is set up, this may be too big for the primary hash table to process. Hence, in certain instances, to deal with the need for such extensive processing, in certain embodiments, the secondary hashing module can be configured to store the information associated with larger seeds. Since the number of seeds requiring extension is a fraction of the total number of seeds, the secondary hash table may be smaller than the primary hash table. However, in other instances, such as to reduce the processing requirements of the module, e.g., to save bits, the known redundant portion of the sequence, e.g., the primary sequence, may be replaced by a preselected variable such as of a predetermined sequence length. In such an instance, since the redundant sequence is already known and identified, it does not need to be digitally represented in its entirety. Rather, in various instances, all that is really needed to be done is to substitute the known, redundant sequence with a known variable sequence, and all that really needs to be looked up are the extension portions, e.g., wings, that have been added to either side of the variable sequence, since those are the only portions of the initial sequence that are non-redundant and new. Hence, in certain instances, the primary sequence may be replaced by a shorter unique identifier code (such as a 24 bit proxy instead of 56 bit representation) and then the extension bases can be added to the proxy, such as a 36 bit extension (e.g., totaling 60 bits) that can then be put into the extend record in the primary table. In a manner such as this, the disadvantages of having too short and/or too long of reads can be minimized and the benefit of having only one or a few look ups in the hash table can be maintained.

[0260] As indicated above, the implementation of the above described hash function may be executed in software and/or hardware. An advantage of implementing the hash module in hardware is that the processes may be accelerated and therefore performed in a much faster manner. For instance, where software may include various instructions for performing one or more of these various functions, the implementation of such instructions often requires data and instructions to be stored and/or fetched and/or read and/or interpreted, such as prior to execution. As indicated above, however, and described in greater detail herein below, a chip can be hardwired to perform these functions without having to fetch, interpret, and/or perform one or more of a sequence of instructions. Rather, the chip may be wired to perform such functions directly. Accordingly, in various aspects, the disclosure is directed to a custom hardwired machine that may be configured such that portions or all of the above described hashing module may be implemented by one or more network circuits, such as integrated circuits hardwired on a chip, such as an FPGA, ASIC or Structured ASIC.

[0261] For instance, in various instances, the hash table index may be constructed and the hash function may be performed on a chip, and in other instances, the hash table index may be generated off of the chip, such as via software run by a host CPU, but once generated it is loaded onto and employed by the chip, such as in running the hash module. In certain instances, the chip may include any suitable number of gigabytes, such as 8 gigabytes, such as 16 gigabytes, such as 32 gigabytes, such as 64 gigabytes, such as about 128 gigabytes. In various instances, the chip may be configurable such that the various processes of the hash module are performed employing only a portion or all the memory resources. For example, where a custom reference genome may be built, a large portion of the memory may be dedicated to storing the hash reference index and/or for storing reads and/or for reserving space for other functional modules to use, such as where 16 gigabytes are dedicated to storing the reads, 8 gigabytes may be dedicated to storing the hash index and another 8 gigabytes may be dedicated to other processing functions. In another example, where 32 gigabytes are dedicated to storing reads, 26 gigabytes may be dedicated for storing the primary hash table, 2.5 gigabytes may be dedicated for storing the secondary table, and 1.5 gigabytes may be dedicated for the reference genome.

[0262] In certain embodiments, the secondary hash table may be constructed so as to have a digital presence that is larger than the primary hash table. For instance, in various instances, the primary hash table can be configured to store hash records of 8 bytes each with 8 records per hash bucket totaling 64 bytes per bucket, and the secondary hash table can be configured to store 16 hash records totaling 128 bytes per bucket. For each hash record containing overflow hash bits matching the same bits of the hash key a possible matching position in the reference genome is reported. For the primary hash table therefore, up to 8 positions may be reported. For the secondary hash table up to 16 positions may be reported.

[0263] Regardless of being implemented in hardware or software, in many instances, it may be useful to structure the hash table to avoid collisions. For instance, there may be multiple seeds that, because of various system artifacts will want to be inserted into the hash table at the same place regardless of whether there is a match there or not. Such instances are termed collisions. Often times, collisions can be avoided, in part, by the way the hash table is structured. Accordingly, in various instances the hash table may be structured so as to avoid collisions, and therefore may be configured to include one or more virtual hash buckets.

[0264] In various instances, the hash table can be structured such that it is represented in an 8 byte, 16 byte, 32 byte, 64 byte, 128 byte format, or the like. But in various exemplary embodiments it may be useful to represent the hash table in a 64 byte format. This may be useful, for instance, where the hash function is to make use of accessing a memory, such as a DRAM, e.g., in a standard DIMM or SODIMM form factor, such as where the minimum burst size is typically 64 bytes. In such an instance, the design of the processor for accessing a given memory will be such that the number of bytes needed to form a bucket in the hash table is also **64**, and therefore a maximized efficiency may be realized. However, if the table were to be structured in a 32 byte format, this would be inefficient because about half the bytes delivered in a burst would contain information not needed by the processor. That would cut the effective byte delivery rate in half. Conversely, if the number of bytes used to form a bucket in the hash table is a multiple of the minimum burst size, e.g., 128, there is no performance penalty as long as the processor actually needs all of the information returned in a single access. Therefore, in instances where the optimal burst size of the memory access is at a given size, e.g., 64 bytes, the hash table can be structured so burst size of the memory is optimally exploited, such as where the bytes allocated for representing bins in the hash table and processed by the mapping function, e.g., 64 bytes, are coincident with the burst size of the memory. Consequently, where the memory bandwidth is a constraint, the hash table can be structured so as to optimally exploit such constraints.

[0265] Further, it is to be noted, that although a record may be crammed into 8 bytes, the hash function can be constructed such that it is not the case that 8 bytes from the table are read so as to process one record, as this could be inefficient. Rather, all 8 records in a bucket can be read at once, or some sub-portion thereof. This may be useful in optimizing the processing speed of the system as, given the architecture described above, it would cost the same time at the same speed to process all 8 records as it would for simply processing **1** record. Accordingly, in certain instances, the mapping module may include a hash table that itself may include one or more subsections, e.g., virtual sections or buckets, wherein each bucket may have 1 or more slots, such as 8 slots, such that one or more different records can be inserted therein such as to manage collisions. However, in certain circumstances, one or more of such buckets may fill up with records, so a means may be provided for storing additional records in other buckets and recording information in the original bucket indicating that the hash table lookup mechanism needs to look further to find a match.

[0266] Hence, in certain instances it may also be useful to employ one or more additional methods such as for managing collisions, one such method may include one or more of linear probing and/or hash chaining. For instance, if it is not known what exactly is being searched in the hash table or a portion thereof, such as in one bucket of the hash table, and the particular bucket is full, then the hash lookup function can be configured such that if one bucket is full and is searched and the desired record not found, then the function can be directed to step to the next bucket, e.g., the +1 bucket, and that bucket can then be checked. In such a manner, all buckets can be searched when looking for a particular record. Such searching, therefore, can be performed sequentially looking through one bucket to another until what is being looked for is found or it becomes clear that it is not going to be found, such as where an empty slot in at least one of the buckets is found. Particularly, where each bucket is filled sequentially, and each bucket is searched according to the sequence of filling, if an empty slot is found, such as when searching sequentially through buckets looking for a particular record, then the empty slot could be indicative of the record not existing, because if it did exist, it would at least have been positioned in the empty slot, if not in the preceding buckets.

[0267] More particularly, where 64 bytes are designated for storing the information in a hash bucket wherein 8 records are contained, upon receiving a fetched bucket, the mapping processor can operate on all 8 records simultaneously to determine which are matches and which are not. For instance, when performing a look up such as of a seed from a read obtained from the sequenced sample DNA against a seed generated from the reference genome, the digital representation of the sample seed can be compared against the reference seeds in all, e.g., **8**, records so as to find a match. In such an instance, several outcomes may result. A direct match may be found. A sample seed may go into the hash table and, in some instances, no match is found, e.g., because it is just not exactly the same as any corresponding seed in the reference, such as because there was a machine or sequencing error with respect to that seed or the read from which it is generated, or because the person has a genetic sequence that is different from the reference genome. Or a the seed may go into the hash table and a plurality of matches may be returned, such where the sample seed matches to 2, 3, 5, 10, 15, 20, or more places in the table. In such an instance, multiple records may be returned all pointing to various different locations in the reference genome where that particular seed matches, the records for these matches may either be in the same bucket, or a multiplicity of buckets may have to be probed to return all of the significant, e.g., match, results.

[0268] In certain instances, such as where space may become a limiting factor in the hash table, e.g., in the hash table buckets, an additional mechanism for resolving collisions and/or for saving space may implemented. For instance, when space becomes limited, such as when more than 8 records need to be stored in a bucket, or when for other instances it is desirable, a hash chaining function may be performed. Hash chaining can involve, for example, replacing a record containing a specific position location in the genomic sequence with a record containing a chain pointer that instead of pointing to a location in the genome points to some other address, e.g., a second bucket in the current hash table e.g. a primary or a secondary hash table. This has the advantage over the linear probing method of enabling the hash lookup mechanism to directly access the bucket containing the desired record rather than checking buckets sequentially in order.

[0269] Such a process may be useful given the system architecture. For instance, the primary seeds being hashed, such as in a primary lookup, are positioned at a given location in the table, e.g., their original position, whereas the seeds being chained are being put in a position that may be different from their original bucket. Hence, as indicated above, a first portion of the digitally represented seed, e.g., about 26 to about 29 bits, can be hashed and may be looked up in a first step. And, in a second step, the remaining about 27 to about 30 bits can be inserted into the hash table, such as in a hash chain, as a means for confirming the first pass. Accordingly, for any seed, its original address bits may be hashed in a first step, and the secondary address bits may be used in a second, confirmation step. Hence, the first portion of the seeds can be inserted into primary record location, and the second portion may be fit into the table in secondary record chain location. And, as indicated above, in various instances, these two different record locations may be positionally separated, such as by a chain format record. Therefore, in any destination bucket of chaining a chain format record may positionally separate the entries/records that are for local primary first bucket accesses and probing and those records that are for the chain.

[0270] Such hash chains can be continued for a multiplicity of lengths. An advantage of such chaining is that where one or more of the buckets include one or more, e.g., 2, 3, 4, 5, 6, or more empty record slots, these empty slots can be used to store the hash chain data. Accordingly, in certain instances, hash chaining may involve starting with an empty slot in one bucket and chaining that slot to another slot in another bucket, where the two buckets may be at remote locations in the hash table. Additional care may be taken to avoid confusion between records placed in a remote bucket as part of a hash chain, and “native” records that hash directly into the same bucket. As usual, the remaining about 27 to about 30 bits of the secondary access key are checked against corresponding about 27 to 30 bits stored in the records placed remotely in the chained bucket, but due to the distant placement of the chained bucket from the original hash bucket, confirming these about 27 to 30 bits would not be enough to guarantee that a matching hash record corresponds to the original seed reaching this bucket by chaining, as opposed to some other seed reaching the same bucket by direct access. (e.g., confirming the about 27 to 30 bits may be a full verification when the about 26 to 29 bits used for hash table addressing are implicitly checked by proximity to the initial hash bucket accessed.)

[0271] To prevent retrieving a wrong hash record without needing to store entire hash keys in the records, a positional system may be used in a chained bucket. Accordingly, a chained bucket must contain a chain continuation format record, which contains a further chain pointer to continue the bucket chain if required; this chain continuation record must appear in a slot of the bucket after all “native” records corresponding to direct hash access, and before all remote records belonging to the chain. During queries, before following any chain pointer, any records appearing after a chain continuation record should be ignored, and after following any chain pointer, any records appearing before a chain continuation record should be ignored.

[0272] For example, where the buckets are about 75%-85% full, 8 buckets may be scanned and only 15-25 slots may be found that can be used, whereas with hash chaining these slots may be found over 2 or 3 or 4 buckets. In such an instance, the number of probe or chain steps required to store a hash record matters because it influences the speed of the system. At run time, if probing is necessary to find the record, a multiplicity of hash look up accesses, e.g., a 64 byte bucket read, may need to be performed which slows the system down. Hash chaining helps to minimize the average number of accesses that have to be performed, because more excess hash records can generally be populated per chained bucket, which can be selected from a wide region, than per probing bucket, which must be sequentially next. Therefore, a given number of excess hash records can typically be populated into a shorter sequence of chained buckets than the necessary sequence of probing buckets, which likewise limits the number of accesses required to locate those excess records in a query. Nevertheless, probing remains valuable for smaller quantities of excess hash records, because probing does not require a bucket slot to be sacrificed for a chain pointer.

[0273] For example, after it has been determined where all the possible matches are for the seeds against the reference genome, it must be determined which out of all the possible locations a given read may match to is in fact the correct position to which it aligns. Hence, after mapping there may be a multiplicity of positions that one or more reads appear to match in the reference genome. Consequently, there may be a plurality of seeds that appear to be indicating the exact same thing, e.g., they may match to the exact same position on the reference, if you take into account the position of the seed in the read.

[0274] The actual alignment, therefore, must be determined for each given read. This determination may be made in several different ways. In one instance, all the reads may be evaluated so as to determine their correct alignment with respect to the reference genome based on the positions indicated by every seed from the read that returned position information during the hash lookup process. However, in various instances, prior to performing an alignment, a seed chain filtering function may be performed on one or more of the seeds.

[0275] For instance, in certain instances, the seeds associated with a given read that appear to map to the same general place as against the reference genome may be aggregated into a single chain that references the same region. All of the seeds associated with one read may be grouped into one or more seed chains such that each seed is a member of only one chain. It is such chain(s) that then cause the read to be aligned to each indicated position in the reference genome. Specifically, in various instances, all the seeds that have the same supporting evidence indicating that they all belong to the same general location(s) in the reference may be gathered together to form one or more chains. The seeds that group together, therefore, or at least appear as they are going to be near one another in the reference genome, e.g., within a certain band, will be grouped into a chain of seeds, and those that are outside of this band will be made into a different chain of seeds.

[0276] Once these various seeds have been aggregated into one or more various seed chains, it may be determined which of the chains actually represents the correct chain to be aligned. This may be done, at least in part, by use of a filtering algorithm that is a heuristic designed to eliminate weak seed chains which are highly unlikely to be the correct one. Generally, longer seed chains, in terms of length spanned within the read, are more likely to be correct, and furthermore, seed chains with more contributing seeds are more likely to be correct. In one example, a heuristic may be applied wherein a relatively strong "superior" seed chain, e.g. long or having many seeds, filters out a relatively weak "inferior" seed chain, e.g. short or having few seeds.

[0277] In one variation, the length of an inferior chain determines a threshold length, e.g. twice as long, such that a superior chain of at least the threshold length can filter it out. In another variation, the seed count of an inferior chain determines a threshold seed count, e.g. five times as many seeds, such that a superior chain of at least the threshold seed count can filter it out. In another variation, the length of an inferior chain determines a threshold seed count, e.g. two times the seed count minus the seed length, such that a superior chain of at least the threshold seed count can filter it out. In some variations, such as when chimeric alignments of reads are desired, only superior seed chains substantially overlapping inferior seed chains within the read may filter them out.

[0278] This process weeds out those seeds that have a low probability of having identified a region of the reference genome where a high quality alignment of the read can be found. It, therefore, may be useful because it reduces the number of alignments that need to be performed for each read thereby accelerating the processing speed and saving time. Accordingly, this process may be employed, in part, as a tuning feature, whereby when greater speed is desired, e.g., high speed mode, more detailed seed chain filtering is performed, and where greater overall accuracy is desired, e.g., enhanced accuracy mode, less seed chain filtering is performed, e.g., all the seed chains are evaluated.

[0279] In various embodiments, seed editing may be performed, such as prior to a seed chain filtering step. For instance, for each read, if all of the seeds of that read are subjected to a mapping function and none of them returned a hit, then there may be a high probability that there was one or more errors in the read, for instance, an error that the sequencer made. In such an instance, an editing function, such as a one-change editing process, e.g., an SNP editing process, can be performed on each seed, such as where a no match outcome was returned. For example, at position X, a one change edit function may instruct that the designated nucleotide be substituted for one of the other 3 nucleotides and it is determined whether a hit, e.g., a match, is obtained by making that change, e.g., a SNP substitution. This one-change editing may be performed in the same manner on every position in the seed and/or on every seed of the read, e.g., substituting each alternative base for each position in the seed. Additionally, where one change is made in one seed, the effects that change would have on every other overlapping seed may be determined in view of that one change.

[0280] Such editing may also be performed for inserts, such as where one of the four nucleotides is added at a given insert position, X, and it is determined if a hit was obtained by making the substitution. This may be done for all four nucleotides and/or for all positions (X, X+1, X+2, X+3, etc.) in the seed and/or all the seeds in the reads. Such editing may also be performed for deletions, such as where one of the four nucleotides is deleted at a given position, X, in the seed, and it is determined if a hit was obtained by making the deletion. This may then be repeated for all positions X+1, X+2, X+3, etc. Such editing, however, can result in a lot of extra processing work and time, such as by requiring a multiplicity of additional lookups, such as 2, or 3, or 4, or 5, or 10, or 50, or 100, or 200, etc. Nevertheless, such extra processing and time may be useful if by such editing an actual hit can be determined, e.g., a match made, where before there was no match. In such an instance, it can then typically be determined that an error was made and further that it was corrected, thereby salvaging the read.

[0281] Additionally, a further heuristic may be employed so as to determine whether an editing function should be performed or not, whereby the algorithm performs a calculation to determine the probability that a hit will be obtained if such editing were to be performed. If a certain threshold probability is met, such as 85% likelihood, then such seed chain editing may be performed. For instance, the system can generate various statistics on the seed chains, such as calculating how many high frequency hits are present and/or how many seed chains contain high frequency hits, and thereby determine if seed chain editing is likely to make a difference in determining matches. For example, if it is determined that there are a large proportion of high frequency hits, then, in such an instance, seed chain editing may be skipped because it is unlikely to make various of the sequences unique enough to give a hit within a reasonable number of hash table look ups, such as 100 or fewer, 50 or fewer, 40 or fewer, 30 or fewer, 20 or fewer, or 10 or fewer. Such statistics can be reviewed and it may then be determined whether to do seed editing or not. For instance, if the statistics show that for any one read, if half the positions show no match, and the others show high frequency matches, then it is probably worth doing seed editing, because where no matches are returned, there is probably an error, but if a lot of high frequency matches are returned it may simply not be worth performing seed editing.

[0282] The outcome from performing one or more of these mapping, filtering, and/or editing functions is a list of reads which includes for each read a list of all the possible locations to where the read may matchup with the reference genome. Hence, a mapping function may be performed so as to quickly determine where the reads of the FASTQ file obtained from the sequencer map to the reference genome, e.g., to where in the whole genome the various reads map. However, if there is an error in any of the reads or a genetic variation, you may not get an exact match to the reference and/or there may be several places one or more reads appear to match. It, therefore, must be determined where the various reads actually align with respect to the genome as a whole.

[0283] Accordingly, after mapping and/or filtering and/or editing, the location positions for a large number of reads have been determined, where for some of the individual reads a multiplicity of location positions have been determined, and it now needs to be determined which out of all the possible locations is in fact the true or most likely location to which the various reads align. Such aligning may be performed by one or more algorithms, such as a dynamic programming algorithm that matches the mapped reads to the reference genome and runs an alignment function thereon.

[0284] An exemplary aligning function compares one or more, e.g., all of the reads, to the reference, such as by placing them in a graphical relation to one another, e.g., such as in a table, e.g., a virtual array or matrix, where the sequence of one of the reference genome or the mapped reads is placed on one dimension or axis, e.g., the horizontal axis, and the other is placed on the opposed dimensions or axis, such as the vertical axis. A conceptual scoring wave front is then passed over the array so as to determine the alignment of the reads with the reference genome, such as by computing alignment scores for each cell in the matrix.

[0285] The scoring wave front represents one or more, e.g., all, the cells of the matrix, or a portion of those cells, which may be scored independently and/or simultaneously according to the rules of dynamic programming applicable in the alignment algorithm, such as Smith-Waterman, and/or Needleman-Wunsch, and/or related algorithms. For example, taking the origin of the matrix (corresponding to the beginning of the read and/or the beginning of a reference window of the conceptual scoring wave front) to be at the top-left corner, first only the top-left cell at coordinates (0,0) of the matrix may be scored, e.g., a 1-cell wave front; next, the two cells to the right and below at coordinates (0,1) and (1,0) may be scored, e.g., a 2-cell wave front; next the three cells at (0,2), (1,1), and (2,0) may be scored, e.g., a 3-cell wave front. These exemplary wave fronts may then extend diagonally in straight lines from bottom-left to top-right, and the motion of the wave front from step to step is diagonally from top-left to bottom-right through the matrix. Alignment scores may be computed sequentially or in other orders, such as by computing all the scores in the top row from left to right, followed by all the scores in the next row from left to right, etc. In this manner the diagonally sweeping diagonal wave front represents an optimal sequence of batches of scores computed simultaneously or in parallel in a series of wave front steps.

[0286] For instance, in one embodiment, a window of the reference genome containing the segment to which a read was mapped is placed on the horizontal axis, and the read is positioned on the vertical axis. In a manner such as this an array or matrix is generated, e.g., a virtual matrix, whereby the nucleotide at each position in the read may be compared with the nucleotide at each position in the reference window. As the wave front passes over the array, all potential ways of aligning the read to the reference window are considered, including if changes to one sequence would be required to make the read match the reference sequence, such as by changing one or more nucleotides of the read to other nucleotides, or inserting one or more new nucleotides into one sequence, or deleting one or more nucleotides from one sequence.

[0287] An alignment score, representing the extent of the changes that would be required to be made to achieve an exact alignment, is generated, wherein this score and/or other associated data may be stored in the given cells of the array. Each cell of the array corresponds to the possibility that the nucleotide at its position on the read axis aligns to the nucleotide at its position on the reference axis, and the score generated for each cell represents the partial alignment terminating with the cell's positions in the read and the reference window. The highest score generated in any cell represents the best overall alignment of the read to the reference window. In various instances, the alignment may be global, where the entire read must be aligned to some portion of the reference window, such as using a Needleman-Wunsch or similar algorithm; or in other instances, the alignment may be local, where only a portion of the read may be aligned to a portion of the reference window, such as by using a Smith-Waterman or similar algorithm.

[0288] The size of the reference window may be any suitable size. For instance, since a typical read may be from about 100 to about 1,000 nucleotides long, the length of the reference window accordingly, in some instances, may be from about 100 to 1,000 nucleotides long or longer. However, in some instances, the length of the reads may be greater, and/or the length of the reference window can be greater such as about 10,000, 25,000, 50,000, 75,000, 100,000, 200,000 nucleotides long or more. It may be advantageous for the reference window to be padded somewhat longer than the read, such as including 32 or 64 or 128 or 200 or even 500 extra nucleotides in the reference window beyond the extremes of the reference genome segment to which the read was mapped, such as to permit insertions and/or deletions near the ends of the read to be fully evaluated. For instance, if only a portion of the read was mapped to a segment of the reference, extra padding may be applied to the reference window corresponding to the unmapped portions of the read, or longer by some factor, such as 10% or 15% or 20% or 25% or even 50% or more, so as to allow the unmapped portions of the read space to fully align to the reference window. In some instances, however, the length of the reference window may be selected to be shorter than the length of the reads, such as where a long portion of the read is not mapped to the reference, such as more or less than 1000 nucleotides at one end of the read, such as in order to focus the alignment on the mapped portion.

[0289] The alignment wave front may be of unlimited length, or limited to any suitable fixed length, or of variable length. For instance, all cells along the entire diagonal line of each wave front step extending fully from one axis to the other axis may be scored. Alternatively, a limited length, such as 64 cells wide, may be scored on each wave front step, such as by tracing a diagonally 64-cell wide band of scored cells through the matrix, and leaving cells outside of this band unscored. In some instances, it may be unnecessary to calculate scores far from a band around the true alignment path, and substantial work may be saved by computing scores only in a limited bandwidth, using a fixed length scoring wave front, as herein described.

[0290] Accordingly, in various instances, an alignment function may be performed, such as on the data obtained from the mapping module. Hence, in various instances, an alignment function may form a module, such as an alignment module, that may form part of a system, e.g., a pipeline, that is used, such as in addition with a mapping module, in a process for determining the actual entire genomic sequence, or a portion thereof, of an individual. For instance, the output returned from the performance of the mapping function, such as from a mapping module, e.g., the list of possibilities as to where one or more or all of the reads maps to one or more positions in one or more reference genomes, may be employed by the alignment function so as to determine the actual sequence alignment of the subject's sequenced DNA.

[0291] Such an alignment function may at times be useful because, as described above, often times, for a variety of different reasons, the sequenced reads do not always match exactly to the reference genome. For instance, there may be an SNP (single nucleotide polymorphism) in one or more of the reads, e.g., a substitution of one nucleotide for another at a single position; there may be an "indel," insertion or deletion of one or more bases along one or more of the read sequences, which insertion or deletion is not present in the reference genome; and/or there may be a sequencing error (e.g., errors in sample prep and/or sequencer read and/or sequencer output, etc.) causing one or more of these apparent variations. Accordingly, when a read varies from the reference, such as by an SNP or indel, this may be because the reference differs from the true DNA sequence sampled, or because the read differs from the true DNA sequence sampled. The problem is to figure out how to correctly align the reads to the reference genome given the fact that in all likelihood the two sequences are going to vary from one another in a multiplicity of different ways.

[0292] Accordingly, in various instances, the input into an alignment function, such as from a mapping function, such as a prefix/suffix tree, or a Burrows/Wheeler transform, or a hash table and/or hash function, may be a list of possibilities as to where one or more reads may match to one or more positions of one or more reference sequences. For instance, for any given read, it may match any number of positions in the reference genome, such as at **1** location or 16, or 32, or 64, or 100, or 500, or 1,000 or more locations where a given read maps to in the genome. However, any individual read was derived, e.g., sequenced, from only one specific portion of the genome. Hence, in order to find the true location from where a given particular read was derived, an alignment function may be performed, e.g., a Smith-Waterman gapped alignment, a Needleman-Wunsch alignment, etc., so as to determine where in the genome one or more of the reads was actually derived, such as by comparing all of the possible locations where a match occurs and determining which of all the possibilities is the most likely location in the genome from which the read was sequenced, on the basis of which location's alignment score is greatest.

[0293] As indicated, typically, an algorithm is used to perform such an alignment function. For example, a Smith-Waterman and/or a Needleman-Wunsch alignment algorithm may be employed to align two or more sequences against one another. In this instance, they may be employed in a manner so as to determine the probabilities that for any given position where the read maps to the reference genome that the mapping is in fact the position from where the read originated. Typically these algorithms are configured so as to be performed by software, however, in various instances, such as herein presented, one or more of these algorithms can be configured so as to be executed in hardware, as described in greater detail herein below.

[0294] In particular, the alignment function operates, at least in part, to align one or more, e.g., all, of the reads to the reference genome despite the presence of one or more portions of mismatches, e.g., SNPs, insertions, deletions, structural artifacts, etc. so as to determine where the reads are likely to fit in the genome correctly. For instance, the one or more reads are compared against the reference genome, and the best possible fit for the read against the genome is determined, while accounting for substitutions and/or indels and/or structural variants. However, to better determine which of the modified versions of the read best fits against the reference genome, the proposed changes must be accounted for, and as such a scoring function may also be performed.

[0295] For instance, a scoring function may be performed, e.g., as part of an overall alignment function, whereby as the alignment module performs its function and introduces one or more changes into a sequence being compared to another, e.g., so as to achieve a better or best fit between the two, for each change that is made so as to achieve the better alignment, a number is detracted from a starting score, e.g., either a perfect score, or a zero starting score, in a manner such that as the alignment is performed the score for the alignment is also determined, such as where matches are detected the score is increased, and for each change introduced a penalty is incurred, and thus, the best fit for the possible alignments can be determined, for example, by figuring out which of all the possible modified reads fits to the genome with the highest score. Accordingly, in various instances, the alignment function may be configured to determine the best combination of changes that need to be made to the read(s) to achieve the highest scoring alignment, which alignment may then be determined to be the correct or most likely alignment.

[0296] In view of the above, there are, therefore, at least two goals that may be achieved from performing an alignment function. One is a report of the best alignment, including position in the reference genome and a description of what changes are necessary to make the read match the reference segment at that position, and the other is the alignment quality score. For instance, in various instances, the output from a the alignment module may be a Compact Idiosyncratic Gapped Alignment Report, e.g., a CIGAR string, wherein the CIGAR string output is a report detailing all the changes that were made to the reads so as to achieve their best fit alignment, e.g., detailed alignment instructions indicating how the query actually aligns with the reference. Such a CIGAR string readout may be useful in further stages of processing so as to better determine that for the given subject's genomic nucleotide sequence, the predicted variations as compared against a reference genome are in fact true variations, and not just due to machine, software, or human error.

[0297] As set forth above, in various embodiments, alignment is typically performed in a sequential manner, wherein the algorithm receives read sequence data, such as from a mapping module, pertaining to a read and one or more possible locations where the read may potentially map to the one or more reference genomes, and further receives genomic sequence data, such as from one or more memories, pertaining to the one or more positions in the one or more reference genomes to which the read may map. In particular, in various embodiments, the mapping module processes the reads, such as from a FASTQ file, and maps each of them to one or more positions in the reference genome to where they may possibly align. The aligner then takes these predicted positions and uses them to align the reads to the reference genome, such as by building a virtual array by which the reads can be compared with the reference genome.

[0298] In performing this function the aligner evaluates each mapped position for each individual read and particularly evaluates those reads that map to multiple possible locations in the reference genome and scores the possibility that each position is the correct position. It then compares the best scores, e.g., the two best scores, and makes a decision as to where the particular read actually aligns. For instance, in comparing the first and second best alignment scores, the aligner looks at the difference between the scores, and if the difference between them is great, then the confidence score that the one with the bigger score is correct will be high. However, where the difference between them is small, e.g., zero, then the confidence score in being able to tell from which of the two positions the read actually is derived is low, and more processing may be useful in being able to clearly determine the true location in the reference genome from where the read is derived. Hence, the aligner in part is looking for the biggest difference between the first and second best confidence scores in making its call that a given read maps to a given location in the reference genome. Ideally, the score of the best possible choice of alignment is significantly greater than the score for the second best alignment for that sequence.

[0299] There are many different ways an alignment scoring methodology may be implemented, for instance, each cell of the array may be scored or a sub-portion of cells may be scored, such as in accordance with the methods disclosed herein. Typically, each alignment match, corresponding to a diagonal step in the alignment matrix, contributes a positive score, such as +1, if the corresponding read and reference nucleotides match; and a negative score, such as −4, if the two nucleotides mismatch. Further, each deletion from the reference, corresponding to a horizontal step in the alignment matrix, contributes a negative score, such as −7, and each insertion into the reference, corresponding to a vertical step in the alignment matrix, contributes a negative score, such as −7.

[0300] In various instances, scoring parameters for nucleotide matches, nucleotide mismatches, insertions, and deletions may have any various positive or negative or zero values. In various instances, these scoring parameters may be modified based on available information. For instance, in certain instances, alignment gaps (insertions or deletions) are penalized by an affine function of the gap length, for example −7 for the first deleted (resp. inserted) nucleotide, but only −1 for each additional deleted (resp. inserted) nucleotide in continuous sequence. In various implementations, affine gap penalties may be achieved by splitting gap (insertion or deletion) penalties into two components, such as a gap open penalty, e.g. −6, applied to the first step in a gap; and a gap extend penalty, e.g. −1, applied to every or further steps in the gap. Affine gap penalties may yield more accurate alignments, such as by letting alignments containing long insertions or deletions achieve appropriately high scores. Further, each lateral move may have the same or different costs, such as the same cost per step, and/or where gaps occur, such gaps can come at a higher or lower costs, such that the cost for lateral movements of the aligner may be less expensive than the costs for gaps. Accordingly, in various embodiments, affine gap scoring may be implemented, however, this can be expensive in software and/or hardware, because it typically requires a plurality, e.g., 3 scores, for each cell to be scored, and hence, in various embodiments affine gap scoring is not implemented.

[0301] In various instances, scoring parameters may also be sensitive to “base quality scores” corresponding to nucleotides in the read. Some sequenced DNA read data, in formats such as FASTQ, may include a base quality score associated with each nucleotide, indicating an estimated probability that the nucleotide is incorrect, e.g. due to a sequencing error. In some read data, base quality scores may indicate the likelihood that an insertion and/or deletion sequencing error is present in or adjacent to each position, or additional quality scores may provide this information separately. More accurate alignments, therefore, may be achieved by making scoring parameters, including any or all of nucleotide match scores, nucleotide mismatch scores, gap (insertion and/or deletion) penalties, gap open penalties, and/or gap extend penalties, vary according to a base quality score associated with the current read nucleotide or position. For example, score bonuses and/or penalties could be made smaller when a base quality score indicates a high probability a sequencing or other error being present. Base quality sensitive scoring may be implemented, for example, using a fixed or configurable lookup-table, accessed using a base quality score, which returns corresponding scoring parameters.

[0302] In a hardware implementation in an integrated circuit, such as an FPGA, ASIC or Structured ASIC, a scoring wave front may be implemented as a linear array of scoring cells, such as 16 cells, or 32 cells, or 64 cells, or 128 cells or the like. Each of the scoring cells may be built of digital logic elements in a wired configuration to compute alignment scores. Hence, for each step of the wave front, for instance, each clock cycle, or some other fixed or variable unit of time, each of the scoring cells, or a portion of the cells, computes the score or scores required for a new cell in the virtual alignment matrix. Notionally, the various scoring cells are considered to be in various positions in the alignment matrix, corresponding to a scoring wave front as discussed herein, e.g., along a straight line extending from bottom-left to top-right in the matrix. As is well understood in the field of digital logic design, the physical scoring cells and their comprised digital logic need not be physically arranged in like manner on the integrated circuit.

[0303] Accordingly, as the wave front takes steps to sweep through the virtual alignment matrix, the notional positions of the scoring cells correspondingly update each cell, for example, notionally “moving” a step to the right, or for example, a step downward in the alignment matrix. All scoring cells make the same relative notional movement, keeping the diagonal wave front arrangement intact. Each time the wave front moves to a new position, e.g., with a vertical downward step, or a horizontal rightward step in the matrix, the scoring cells arrive in new notional positions, and compute alignment scores for the virtual alignment matrix cells they have entered.

[0304] In such an implementation, neighboring scoring cells in the linear array are coupled to communicate query (read) nucleotides, reference nucleotides, and previously calculated alignment scores. The nucleotides of the reference window may be fed sequentially into one end of the wave front, e.g., the top-right scoring cell in the linear array, and may shift from there sequentially down the length of the wave front, so that at any given time, a segment of reference nucleotides equal in length to the number of scoring cells is present within the cells, one successive nucleotide in each successive scoring cell.

[0305] Accordingly, each time the wave front steps horizontally, another reference nucleotide is fed into the top-right cell, and other reference nucleotides shift down-left through the wave front. This shifting of reference nucleotides may be the underlying reality of the notional movement of the wave front of scoring cells rightward through the alignment matrix. Hence, the nucleotides of the read may be fed sequentially into the opposite end of the wave front, e.g. the bottom-left scoring cell in the linear array, and shift from there sequentially up the length of the wave front, so that at any given time, a segment of query nucleotides equal in length to the number of scoring cells is present within the cells, one successive nucleotide in each successive scoring cell.

[0306] Likewise, each time the wave front steps vertically, another query nucleotide is fed into the bottom-left cell, and other query nucleotides shift up-right through the wave front. This shifting of query nucleotides is the underlying reality of the notional movement of the wave front of scoring cells downward through the alignment matrix. Accordingly, by commanding a shift of reference nucleotides, the wave front may be moved a step horizontally, and by commanding a shift of query nucleotides, the wave front may be moved a step vertically. Accordingly, to produce generally diagonal wave front movement, such as to follow a typical alignment of query and reference sequences without insertions or deletions, wave front steps may be commanded in alternating vertical and horizontal directions.

[0307] Accordingly, neighboring scoring cells in the linear array may be coupled to communicate previously calculated alignment scores. In various alignment scoring algorithms, such as a Smith-Waterman or Needleman-Wunsch, or such variant, the alignment score(s) in each cell of the virtual alignment matrix may be calculated using previously calculated scores in other cells of the matrix, such as the three cells positioned immediately to the left of the current cell, above the current cell, and diagonally up-left of the current cell. When a scoring cell calculates new score(s) for another matrix position it has entered, it must retrieve such previously calculated scores corresponding to such other matrix positions. These previously calculated scores may be obtained from storage of previously calculated scores within the same cell, and/or from storage of previously calculated scores in the one or two neighboring scoring cells in the linear array. This is because the three contributing score positions in the virtual alignment matrix (immediately left, above, and diagonally up-left) would have been scored either by the current scoring cell, or by one of its neighboring scoring cells in the linear array.

[0308] For instance, the cell immediately to the left in the matrix would have been scored by the current scoring cell, if the most recent wave front step was horizontal (rightward), or would have been scored by the neighboring cell down-left in the linear array, if the most recent wave front step was vertical (downward). Similarly, the cell immediately above in the matrix would have been scored by the current scoring cell, if the most recent wave front step was vertical (downward), or would have been scored by the neighboring cell up-right in the linear array, if the most recent wave front step was horizontal (rightward). Similarly, the cell diagonally up-left in the matrix would have been scored by the current scoring cell, if the most recent two wave front steps were in different directions, e.g., down then right, or right then down, or would have been scored by the neighboring cell up-right in the linear array, if the most recent two wave front steps were both horizontal (rightward), or would have been scored by the neighboring cell down-left in the linear array, if the most recent two wave front steps were both vertical (downward).

[0309] Accordingly, by considering information on the last one or two wave front step directions, a scoring cell may select the appropriate previously calculated scores, accessing them within itself, and/or within neighboring scoring cells, utilizing the coupling between neighboring cells. In a variation, scoring cells at the two ends of the wave front may have their outward score inputs hard-wired to invalid, or zero, or minimum-value scores, so that they will not affect new score calculations in these extreme cells.

[0310] A wave front being thus implemented in a linear array of scoring cells, with such coupling for shifting reference and query nucleotides through the array in opposing directions, in order to notionally move the wave front in vertical and horizontal steps, and coupling for accessing scores previously computed by neighboring cells in order to compute alignment score(s) in new virtual matrix cell positions entered by the wave front, it is accordingly possible to score a band of cells in the virtual matrix, the width of the wave front, such as by commanding successive steps of the wave front to sweep it through the matrix. For a new read and reference window to be aligned, therefore, the wave front may begin positioned inside the scoring matrix, or, advantageously, may gradually enter the scoring matrix from outside, beginning e.g., to the left, or above, or diagonally left and above the top-left corner of the matrix.

[0311] For instance, the wave front may begin with its top-left scoring cell positioned just left of the top-left cell of the virtual matrix, and the wave front may then sweep rightward into the matrix by a series of horizontal steps, scoring a horizontal band of cells in the top-left region of the matrix. When the wave front reaches a predicted alignment relationship between the reference and query, or when matching is detected from increasing alignment scores, the wave front may begin to sweep diagonally down-right, by alternating vertical and horizontal steps, scoring a diagonal band of cells through the middle of the matrix. When the bottom-left wave front scoring cell reaches the bottom of the alignment matrix, the wave front may begin sweeping rightward again by successive horizontal steps, until some or all wave front cells sweep out of the boundaries of the alignment matrix, scoring a horizontal band of cells in the bottom-right region of the matrix.

[0312] In a variation, increased efficiency may be obtained from the alignment wave front by sharing its scoring cells between two successive alignment operations. A next alignment matrix having been established in advance, as the top-right portion of the wave front exits the bottom-right region of the current alignment matrix, it may enter, immediately, or after crossing a minimum gap such as one cell or three cells, the top-right region of the next alignment matrix. In this manner, the horizontal wave front sweep out of one alignment matrix can be the same motion as the horizontal wave front sweep into the next alignment matrix. Doing this may include the reference and query bases of the next alignment to be fed into those scoring cells crossing into the next alignment matrix, and can reduce the average time consumed per alignment by the time to execute a number of wave front steps almost equal to the number of alignment cells in the wave front, e.g., such as 64 or 63 or 61 steps, which may take e.g. 64 or 63 or 61 clock cycles.

[0313] The number of scoring cells in an implementation of an alignment wave front may be selected to balance various factors, including alignment accuracy, maximum insertion and deletion length, area, cost, and power consumption of the digital logic, clock frequency of the aligner logic, and performance of the overall integrated circuit. A long wave front is desirable for good alignment accuracy, especially because a wave front of N cells can align across indels approximately N nucleotides long, or slightly shorter. But a longer wave front costs more logic, which consumes more power. Further, a longer wave front can increase wire routing complexity and delays on the integrated circuit, leading to lower maximum clock frequencies, reducing net aligner performance. Further still, if an integrated circuit has a limited size or power consumption, using a longer wave front may require less logic to be implemented on the IC elsewhere, such as replicating fewer entire wave fronts, or other aligner or mapper logic components, this decreasing net performance of the IC. In one particular embodiment, 64 scoring cells in the wave front may give an acceptable balance of these factors.

[0314] Accordingly, where the wave front is X, e.g., 64 scoring cells wide, the scored band in the alignment matrix will likewise be 64 cells wide (measured diagonally). The matrix cells outside of this band do not necessarily need to be processed nor their scores calculated, provided that the optimal (best-scoring) alignment path through the matrix stays within the scored band. In a relatively small matrix, therefore, used to align relatively short reads, e.g., 100 nucleotide or 250 nucleotide reads, this may be a safe assumption, such as if the wave front sweeps a perfect diagonal along the predicted aligned position of the read.

[0315] However, in some instances, such as in a large alignment matrix used to align long reads, e.g., 1000 or 10,000 or 100,000 nucleotides, there may be a substantial risk of accumulated indels causing the true alignment to deviate from a perfect diagonal, sufficiently far in aggregate that it may escape the scored band. In such instances, it may be useful to steer the wave front so that the highest set of scores will be near the center of the wave front. Consequently, as the wave front performs its sweep, if the highest scores start to move one way or the other, e.g., left to right, the wave front is shifted over to track this move. For instance, if the highest scores are observed in scoring cells substantially up-right from the center of the wave front, the wave front may be steered some distance straight rightward by successive horizontal steps, until the highest scores return near the center of the wave front.

[0316] Accordingly, an automatic steering mechanism may be implemented in the wave front control logic, to determine a steering target position within the length of the wave front, based on current and past scores observed in the wave front scoring cells, and to steer the wave front toward this target if it is off-center. More particularly, the position of the maximum score in the most recently scored wave front position may be used as a steering target. This is an effective method in some instances. In some instances, however, the maximum score position may be a poor steering target. For instance, with some combinations of alignment scoring parameters, when a long indel commences, and scores accordingly begin to decline, a pattern of two higher-score peaks with a lower-score valley between them can form along the wave front, the two peaks drifting apart as the indel continues.

[0317] Because it cannot be easily determined whether the event in progress is an insertion or a deletion, it is important for the wave front to track diagonally until successful matching commences again, either some distance to the right for a deletion, or some distance downward for an insertion. But if two spreading score peaks form, one of them is likely to be slightly higher than the other, and could pull the automatic steering in that direction, causing the wave front to lose the alignment if the actual indel was in the other direction. A more robust method, therefore, may be to subtract a delta value from the maximum observed wave front score to determine a threshold score, identify the two extreme scoring cells at least equal to this threshold score, and use the midpoint between these extreme cells as the steering target. This will tend to guide diagonally between a two-peak score pattern. Other steering criteria can readily be applied, however, which serve to keep higher scores near the center of the wave front. If there is a delayed reaction between obtaining scores from wave front scoring cells and making a corresponding steering decision, hysteresis can advantageously be applied to compensate for steering decisions made in the intervening time, to avoid oscillating patterns of automatic wave front steering.

[0318] One or more of such alignment procedures may be performed by any suitable alignment algorithm, such as a Needleman-Wunsch alignment algorithm and/or a Smith-Waterman alignment algorithm that may have been modified to accommodate the functionality herein described. In general both of these algorithms and those like them basically perform, in some instances, in a similar manner. For instance, as set forth above, these alignment algorithms typically build the virtual array in a similar manner such that, in various instances, the horizontal top boundary may be configured to represent the genomic reference sequence, which may be laid out across the top row of the array according to its base pair composition. Likewise, the vertical boundary may be configured to represent the sequenced and mapped query sequences that have been positioned in order, downwards along the first column, such that their nucleotide sequence order is generally matched to the nucleotide sequence of the reference to which they mapped. The intervening cells may then be populated with scores as to the probability that the relevant base of the query at a given position, is positioned at that location relative to the reference. In performing this function, a swath may be moved diagonally across the matrix populating scores within the intervening cells and the probability for each base of the query being in the indicated position may be determined.

[0319] With respect to a Needleman-Wunsch alignment function, which generates optimal global (or semi-global) alignments, aligning the entire read sequence to some segment of the reference genome, the wave front steering may be configured such that it typically sweeps all the way from the top edge of the alignment matrix to the bottom edge. When the wave front sweep is complete, the maximum score on the bottom edge of the alignment matrix (corresponding to the end of the read) is selected, and the alignment is back-traced to a cell on the top edge of the matrix (corresponding to the beginning of the read). In various of the instances disclosed herein, the reads can be any length long, can be any size, and there need not be extensive read parameters as to how the alignment is performed, e.g., in various instances, the read can be as long as a chromosome. In such an instance, however, the memory size and chromosome length may be limiting factor.

[0320] With respect to a Smith-Waterman algorithm, which generates optimal local alignments, aligning the entire read sequence or part of the read sequence to some segment of the reference genome, this algorithm may be configured for finding the best scoring possible based on a full or partial alignment of the read. Hence, in various instances, the wave front-scored band may not extend to the top and/or bottom edges of the alignment matrix, such as if a very long read had only seeds in its middle mapping to the reference genome, but commonly the wave front may still score from top to bottom of the matrix. Local alignment is typically achieved by two adjustments. First, alignment scores are never allowed to fall below zero (or some other floor), and if a cell score otherwise calculated would be negative, a zero score is substituted, representing the start of a new alignment. Second, the maximum alignment score produced in any cell in the matrix, not necessarily along the bottom edge, is used as the terminus of the alignment. The alignment is backtraced from this maximum score up and left through the matrix to a zero score, which is used as the start position of the local alignment, even if it is not on the top row of the matrix.

[0321] In view of the above, there are several different possible pathways through the virtual array. In various embodiments, the wave front starts from the upper left corner of the virtual array, and moves downwards towards identifiers of the maximum score. For instance, the results of all possible aligns can be gathered, processed, correlated, and scored to determine the maximum score. When the end of a boundary or the end of the array has been reached and/or a computation leading to the highest score for all of the processed cells is determined (e.g., the overall highest score identified) then a backtrace may be performed so as to find the pathway that was taken to achieve that highest score.

[0322] For example, a pathway that leads to a predicted maximum score may be identified, and once identified an audit may be performed so as to determine how that maximum score was derived, for instance, by moving backwards following the best score alignment arrows retracing the pathway that led to achieving the identified maximum score, such as calculated by the wave front scoring cells. This backwards reconstruction or backtrace involves starting from a determined maximum score, and working backward through the previous cells navigating the path of cells having the scores that led to achieving the maximum score all the way up the table and back to an initial boundary, such as the beginning of the array, or a zero score in the case of local alignment.

[0323] During a backtrace, having reached a particular cell in the alignment matrix, the next backtrace step is to the neighboring cell, immediately leftward, or above, or diagonally up-left, which contributed the best score that was selected to construct the score in the current cell. In this manner, the evolution of the maximum score may be determined, thereby figuring out how the maximum score was achieved. The backtrace may end at a corner, or an edge, or a boundary, or may end at a zero score, such as in the upper left hand corner of the array. Accordingly, it is such a back trace that identifies the proper alignment and thereby produces the CIGAR strand readout, e.g., 3M, 2D, 8M, 4I, 16M, etc., that represents how the sample genomic sequence derived from the individual, or a portion thereof, matches to, or otherwise aligns with, the genomic sequence of the reference DNA.

[0324] Accordingly, once it has been determined where each read is mapped, and further determined where each read is aligned, e.g., each relevant read has been given a position and a quality score reflecting the probability that the position is the correct alignment, such that the nucleotide sequence for the subject's DNA is known, then the order of the various reads and/or genomic nucleic acid sequence of the subject may be verified, such as by performing a back trace function moving backwards up through the array so as to determine the identity of every nucleic acid in its proper order in the sample genomic sequence. Consequently, in some aspects, the present disclosure is directed to a back trace function, such as is part of an alignment module that performs both an alignment and a back trace function, such as a module that may be part of a pipeline of modules, such as a pipeline that is directed at taking raw sequence read data, such as form a genomic sample form an individual, and mapping and/or aligning that data, which data may then be sorted.

[0325] To facilitate the backtrace operation, it is useful to store a scoring vector for each scored cell in the alignment matrix, encoding the score-selection decision. For classical Smith-Waterman and/or Needleman-Wunsch scoring with linear gap penalties, the scoring vector can encode four possibilities, which may optionally be stored as a 2-bit integer from 0 to 3, for example: 0=new alignment (null score selected); 1=vertical alignment (score from the cell above selected, modified by gap penalty); 2=horizontal alignment (score from the cell to the left selected, modified by gap penalty); 3=diagonal alignment (score from the cell up and left selected, modified by nucleotide match or mismatch score). Optionally, the computed score(s) for each scored matrix cell may also be stored (in addition to the maximum achieved alignment score which is standardly stored), but this is not generally necessary for backtrace, and can consume large amounts of memory. Performing backtrace then becomes a matter of following the scoring vectors; when the backtrace has reached a given cell in the matrix, the next backtrace step is determined by the stored scoring vector for that cell, e.g.: 0=terminate backtrace; 1=backtrace upward; 2=backtrace leftward; 3=backtrace diagonally up-left.

[0326] Such scoring vectors may be stored in a two-dimensional table arranged according to the dimensions of the alignment matrix, wherein only entries corresponding to cells scored by the wave front are populated. Alternatively, to conserve memory, more easily record scoring vectors as they are generated, and more easily accommodate alignment matrices of various sizes, scoring vectors may be stored in a table with each row sized to store scoring vectors from a single wave front of scoring cells, e.g. 128 bits to store 64 2-bit scoring vectors from a 64-cell wave front, and a number of rows equal to the maximum number of wave front steps in an alignment operation.

[0327] Additionally, for this option, a record may be kept of the directions of the various wavefront steps, e.g., storing an extra, e.g., 129th, bit in each table row, encoding e.g., 0 for vertical wavefront step preceding this wavefront position, and 1 for horizontal wavefront step preceding this wavefront position. This extra bit can be used during backtrace to keep track of which virtual scoring matrix positions the scoring vectors in each table row correspond to, so that the proper scoring vector can be retrieved after each successive backtrace step. When a backtrace step is vertical or horizontal, the next scoring vector should be retrieved from the previous table row, but when a backtrace step is diagonal, the next scoring vector should be retrieved from two rows previous, because the wavefront had to take two steps to move from scoring any one cell to scoring the cell diagonally right-down from it.

[0328] In the case of affine gap scoring, scoring vector information may be extended, e.g. to 4 bits per scored cell. In addition to the e.g. 2-bit score-choice direction indicator, two 1-bit flags may be added, a vertical extend flag, and a horizontal extend flag. According to the methods of affine gap scoring extensions to Smith-Waterman or Needleman-Wunsch or similar alignment algorithms, for each cell, in addition to the primary alignment score representing the best-scoring alignment terminating in that cell, a 'vertical score' should be generated, corresponding to the maximum alignment score reaching that cell with a final vertical step, and a 'horizontal score' should be generated, corresponding to the maximum alignment score reaching that cell with a final horizontal step; and when computing any of the three scores, a vertical step into the cell may be computed either using the primary score from the cell above minus a gap-open penalty, or using the vertical score from the cell above minus a gap-extend penalty, whichever is greater; and a horizontal step into the cell may be computed either using the primary score from the cell to the left minus a gap-open penalty, or using the horizontal score from the cell to the left minus a gap-extend penalty, whichever is greater. In cases where the vertical score minus a gap extend penalty is selected, the vertical extend flag in the scoring vector should be set, e.g. '1', and otherwise it should be unset, e.g. '0'. In cases when the horizontal score minus a gap extend penalty is selected, the horizontal extend flag in the scoring vector should be set, e.g. '1', and otherwise it should be unset, e.g. '0'. During backtrace for affine gap scoring, any time backtrace takes a vertical step upward from a given cell, if that cell's scoring vector's vertical extend flag is set, the following backtrace step must also be vertical, regardless of the scoring vector for the cell above. Likewise, any time backtrace takes a horizontal step leftward from a given cell, if that cell's scoring vector's horizontal extend flag is set, the following backtrace step must also be horizontal, regardless of the scoring vector for the cell to the left.

[0329] Accordingly, such a table of scoring vectors, e.g. 129 bits per row for 64 cells using linear gap scoring, or 257 bits per row for 64 cells using affine gap scoring, with some number NR of rows, is adequate to support backtrace after concluding alignment scoring where the scoring wavefront took NR steps or fewer. For example, when aligning 300-nucleotide reads, the number of wavefront steps required may always be less than 1024, so the table may be 257×1024 bits, or approximately 32 kilobytes, which in many cases may be a reasonable local memory inside the IC. But if very long reads are to be aligned, e.g. 100,000 nucleotides, the memory requirements for scoring vectors may be quite large, e.g. 8 megabytes, which may be very costly to include as local memory inside the IC. For such support, scoring vector information may be recorded to bulk memory outside the IC, e.g. DRAM, but then the bandwidth requirements, e.g. 257 bits per clock cycle per aligner module, may be excessive, which may bottleneck and dramatically reduce aligner performance.

[0330] Accordingly, it is desirable to have a method for disposing of scoring vectors before completing alignment, so their storage requirements can be kept bounded, e.g. to perform incremental backtraces, generating incremental partial CIGAR strings for example, from early portions of an alignment's scoring vector history, so that such early portions of the scoring vectors may then be discarded. The challenge is that the backtrace is supposed to begin in the alignment's terminal, maximum scoring cell, which unknown until the alignment scoring completes, so any backtrace begun before alignment completes may begin from the wrong cell, not along the eventual final optimal alignment path.

[0331] Accordingly, a method is given for performing incremental backtrace from partial alignment information, e.g. comprising partial scoring vector information for alignment matrix cells scored so far. From a currently completed alignment boundary, e.g., a particular scored wave front position, backtrace is initiated from all cell positions on the boundary. Such backtrace from all boundary cells may be performed sequentially, or advantageously, especially in a hardware implementation, all the backtraces may be performed together. It is not necessary to extract alignment notations, e.g., CIGAR strings, from these multiple backtraces; only to determine what alignment matrix positions they pass through during the backtrace. In an implementation of simultaneous backtrace from a scoring boundary, a number of 1-bit registers may be utilized, corresponding to the number of alignment cells, initialized e.g., all to '1's, representing whether any of the backtraces pass through a corresponding position. For each step of simultaneous backtrace, scoring vectors corresponding to all the current '1's in these registers, e.g. from one row of the scoring vector table, can be examined, to determine a next backtrace step corresponding to each '1' in the registers, leading to a following position for each '1' in the registers, for the next simultaneous backtrace step.

[0332] Importantly, it is easily possible for multiple '1's in the registers to merge into common positions, corresponding to multiple of the simultaneous backtraces merging together onto common backtrace paths. Once two or more of the simultaneous backtraces merge together, they remain merged indefinitely, because henceforth they will utilize scoring vector information from the same cell. It has been observed, empirically and for theoretical reasons, that with high probability, all of the simultaneous backtraces merge into a singular backtrace path, in a relatively small number of backtrace steps, which e.g. may be a small multiple, e.g. 8, times the number of scoring cells in the wavefront. For example, with a 64-cell wavefront, with high probability, all backtraces from a given wavefront boundary merge into a single backtrace path within 512 backtrace steps. Alternatively, it is also possible, and not uncommon, for all backtraces to terminate within the number, e.g. 512, of backtrace steps.

[0333] Accordingly, the multiple simultaneous backtraces may be performed from a scoring boundary, e.g. a scored wavefront position, far enough back that they all either terminate or merge into a single backtrace path, e.g. in 512 backtrace steps or fewer. If they all merge together into a singular backtrace path, then from the location in the scoring matrix where they merge, or any distance further back along the singular backtrace path, an incremental backtrace from partial alignment information is possible. Further backtrace from the merge point, or any distance further back, is commenced, by normal singular backtrace methods, including recording the corresponding alignment notation, e.g., a partial CIGAR string. This incremental backtrace, and e.g. partial CIGAR string, must be part of any possible final backtrace, and e.g. full CIGAR string, that would result after alignment completes, unless such final backtrace would terminate before reaching the scoring boundary where simultaneous backtrace began, because if it reaches the scoring boundary, it must follow one of the simultaneous backtrace paths, and merge into the singular backtrace path, now incrementally extracted.

[0334] Therefore, all scoring vectors for the matrix regions corresponding to the incrementally extracted backtrace, e.g., in all table rows for wave front positions preceding the start of the extracted singular backtrace, may be safely discarded. When the final backtrace is performed from a maximum scoring cell, if it terminates before reaching the scoring boundary (or alternatively, if it terminates before reaching the start of the extracted singular backtrace), the incremental alignment notation, e.g. partial CIGAR string, may be discarded. If the final backtrace continues to the start of the extracted singular backtrace, its alignment notation, e.g., CIGAR string, may then be grafted onto the incremental alignment notation, e.g., partial CIGAR string.

[0335] Furthermore, in a very long alignment, the process of performing a simultaneous backtrace from a scoring boundary, e.g., scored wave front position, until all backtraces terminate or merge, followed by a singular backtrace with alignment notation extraction, may be repeated multiple times, from various successive scoring boundaries. The incremental alignment notation, e.g. partial CIGAR string, from each successive incremental backtrace may then be grafted onto the accumulated previous alignment notations, unless the new simultaneous backtrace or singular backtrace terminates early, in which case accumulated previous alignment notations may be discarded. The eventual final backtrace likewise grafts its alignment notation onto the most recent accumulated alignment notations, for a complete backtrace description, e.g. CIGAR string.

[0336] Accordingly, in this manner, the memory to store scoring vectors may be kept bounded, assuming simultaneous backtraces always merge together in a bounded number of steps, e.g. 512 steps. In rare cases where simultaneous backtraces fail to merge or terminate in the bounded number of steps, various exceptional actions may be taken, including failing the current alignment, or repeating it with a higher bound or with no bound, perhaps by a different or traditional method, such as storing all scoring vectors for the complete alignment, such as in external DRAM. In a variation, it may be reasonable to fail such an alignment, because it is extremely rare, and even rarer that such a failed alignment would have been a best-scoring alignment to be used in alignment reporting.

[0337] In an optional variation, scoring vector storage may be divided, physically or logically, into a number of distinct blocks, e.g. 512 rows each, and the final row in each block may be used as a scoring boundary to commence a simultaneous backtrace. Optionally, a simultaneous backtrace may be required to terminate or merge within the single block, e.g. 512 steps. Optionally, if simultaneous backtraces merge in fewer steps, the merged backtrace may nevertheless be continued through the whole block, before commencing an extraction of a singular backtrace in the previous block. Accordingly, after scoring vectors are fully written to block N, and begin writing to block N+1, a simultaneous backtrace may commence in block N, followed by a singular backtrace and alignment notation extraction in block N−1. If the speed of the simultaneous backtrace, the singular backtrace, and alignment scoring are all similar or identical, and can be performed simultaneously, e.g., in parallel hardware in an IC, then the singular backtrace in block N−1 may be simultaneous with scoring vectors filling block N+2, and when block N+3 is to be filled, block N−1 may be released and recycled.

[0338] Thus, in such an implementation, a minimum of 4 scoring vector blocks may be employed, and may be utilized cyclically. Hence, the total scoring vector storage for an aligner module may be 4 blocks of 257×512 bits each, for example, or approximately 64 kilobytes. In a variation, if the current maximum alignment score corresponds to an earlier block than the current wavefront position, this block and the previous block may be preserved rather than recycled, so that a final backtrace may commence from this position if it remains the maximum score; having an extra 2 blocks to keep preserved in this manner brings the minimum, e.g., to 6 blocks. In another variation, to support overlapped alignments, the scoring wave front crossing gradually from one alignment matrix to the next as described above, additional blocks, e.g. 1 or 2 additional blocks, may be utilized, e.g., 8 blocks total, e.g., approximately 128 kilobytes. Accordingly, if such a limited number of blocks, e.g., 4 blocks or 8 blocks, is used cyclically, alignment and backtrace of arbitrarily long reads is possible, e.g., 100,000 nucleotides, or an entire chromosome, without the use of external memory for scoring vectors.

[0339] As described above, certain regions of DNA are genes, which encode for proteins or functional RNA. Each gene exists on a single strand of the double-stranded DNA double-helix, often as a series of exons (coding segments) separated by introns (non-coding segments). Some genes have only a single exon, but most have several exons (separated by introns), and some have hundreds of exons or thousands of exons. Exons are commonly a few hundred nucleotides long, but may be as short as only a few nucleotides or as long as tens or hundreds of thousands. Introns are commonly thousands of nucleotides long, and some exceed a million nucleotides.

[0340] A gene may be transcribed by RNA polymerase enzymes into messenger RNA (mRNA) or other types of RNA. The immediate RNA transcript is a single-stranded copy of the gene, except that DNA thymine (T) bases are transcribed into RNA Uracil (U) bases. But immediately after this copy is produced, the intron-copies are usually spliced out by spliceosomes, leaving the exon-copies concatenated together at “splice junctions” (which are not thereafter directly evident). RNA splicing does not always occur in the same way. Sometimes one or more exons are spliced out, and sometimes splice junctions do not fall on the most common intron/exon boundaries. Thus, a single gene can produce multiple different transcribed RNA segments, a process sometimes known as alternative splicing.

[0341] Spliced mRNA is transported (in eukaryotes) out of the cellular nucleus to a ribosome, which decodes it into a protein, each group of three RNA nucleotides (e.g., codon) codes for one amino acid. In this manner, genes in DNA serve as original instructions for the manufacture of proteins.

[0342] RNA splicing tends to occur at consistent exon/intron boundaries, which are characterized by typical sequence content, especially near the ends of the introns. In particular, the first two and last two bases of an intron, called an intron motif, follow one of only 3 sequences, the “canonical” intron motifs, the vast majority of the time (roughly

99.9%). The most common canonical intron motif is "GT/AG", meaning the first two bases of the intron are 'G', 'T', and the last two bases are 'A', 'G'. The GT/AG motif occurs roughly 98.8% of the time. The other canonical intron motifs are GC/AG, occurring roughly 1.0% of the time, and AT/AC, occurring roughly 0.1% of the time. These canonical motifs and their prevalence rates are reasonably consistent across species, but may not be universal.

[0343] Not all genes are transcribed, and those that are may be transcribed at different rates. Many factors can influence whether a given gene is transcribed into RNA, and how often. Some of these factors are inherited, some vary by cell specialization from one tissue to another, and some vary over time with environmental conditions or diseases or genetic mutational artifacts. Therefore, two cells with exactly the same DNA may produce quite different types and quantities of proteins and functional RNA. Because of this, sequencing (reading) the RNA present in one or more cells provides different information from sequencing the DNA. A more complete picture of cellular condition and activity is provided by combining DNA sequencing and RNA sequencing.

[0344] Whole-transcriptome RNA sequencing is commonly performed by first selecting the target RNA, such as protein-coding RNA, then using reverse-transcriptase enzymes to convert the RNA segments back into strands of complementary DNA (cDNA). This DNA can be amplified with polymerase chain reaction (PCR) and/or fragmented into a desired distribution of sequence lengths. Then, the DNA fragments are sequenced with a DNA/RNA sequencer, such as a "shotgun" next-generation sequencer.

[0345] The resulting DNA reads are either reverse-complemented or forward copies of the original RNA strands, except that 'U's are replaced again with 'T's. With some library preparation and sequencing protocols, the orientation of the sequenced DNA strands relative to the original RNA may be maintained or flagged; but in common protocols, approximately 50% of the sequenced DNA will be reverse-complemented relative to the original RNA, with no direct indication of orientation (although there are indirect indications).

[0346] The DNA reads from RNA-seq protocols are different from whole-genome or whole-exome DNA sequencing in other ways. First, aside from contaminants, only transcribed RNA gets sequenced, so non-coding DNA and inactive genes do not generally appear. Second, the quantity of sequenced reads corresponding to various genes is related to the biological transcription rates of those genes. Third, due to intron splicing, the RNA-seq reads tend to skip over intron (non-coding) segments within genes.

[0347] RNA-seq reads are usually processed quite differently from DNA reads. Although both types of reads are typically mapped and aligned to a reference genome, the techniques of DNA and RNA mapping and alignment may differ (as may be seen below). After mapping and alignment, reads are commonly sorted by their mapped reference positions, for both DNA and RNA. Duplicate marking, which is optional for DNA processing, is not commonly used for RNA-seq data.

[0348] After this, DNA reads are commonly processed by a variant caller, to identify differences between the sampled DNA and the reference genome. RNA-seq reads are not commonly used for variant calling, although this is occasionally done. More commonly, aligned and sorted RNA reads are analyzed to determine which genes were expressed in what relative quantities, and/or which of various alternatively-spliced transcripts were produced in what relative quantities. This analysis commonly involves counting how many reads align to various genes, exons, etc., and may also involve transcript assembly (reference-based or de novo) to infer from relatively short RNA-seq reads how the longer RNA transcripts were likely spiced from the DNA.

[0349] Gene, exon, or transcript expression analysis is often extended to differential expression analysis, in which RNA-seq data from multiple samples, often from two or more different classes (sub-populations or phenotypes), is compared to quantify to what extent the genes, exon, or transcripts were expressed differently in different classes. This can include calculating the likelihood of a "null hypothesis" that corresponding expression levels were the same in the different classes, as well as estimating the "fold change" in expression between the samples, e.g. an 8- or 10-fold difference.

[0350] For many applications of DNA and/or RNA sequencing, an early processing stage is mapping and aligning reads to a reference genome. Normally, a DNA-oriented reference genome is used for both DNA and RNA sequencing, with 'T's not 'U's present, especially considering RNA-seq usually involved reverse-transcription into cDNA before sequencing. In the case of RNA-seq, as with whole-exome sequencing for that matter, the reference genome could conceivably be restricted to known coding regions, or to regions near coding DNA. However, it is common practice to map and align to a whole reference genome for the sampled organism.

[0351] The biggest difference required in an RNA-capable mapper/aligner is the ability to handle splice junctions. Because RNA-seq reads correspond to segments of transcribed and spliced RNA, commonly a read crosses one or more splice junctions. With respect to the DNA-oriented reference genome, this means a first portion of the read came from, and should map to, a first exon, a second portion of the read should map to a second exon, and so forth. For example, in a 100-base read, the first 40 bases may come from an exon at Chromosome 3 offset 2,345,000, and the remaining 60 bases may come from another exon 100,000 bases away, starting at chromosome offset 2,445,040. The alignment for such a read may be represented with mapping position Chr3:2345000, and alignment CIGAR string "40M100000N60M", in which the "40M" and "60M" represent the portions aligned to respective exons, and the "100000N" represents a 100,000 base intron, these 100,000 reference bases being skipped by the read alignment. (Abstractly, this CIGAR string can be seen as equivalent to "40M100000D60M", where "100000D" represents a 100,000 base deletion from the reference, but it is customary to represent assumed spliced-out introns with 'N' versus deletions from assumed mutations or sequencing errors with 'D'.)

[0352] A practical difference between 'N' (intron) and 'D' (deletion) CIGAR events relates to their typical lengths. Deletion events are rarely longer than 50 bases, and as such are usefully discovered and precisely positioned using Smith-Waterman or similar sequence alignment algorithms and methods, as described herein. Introns are often many thousands of bases long, or even a million bases or more, and can be burdensome to use Smith-Waterman type aligners to detect such long alignment gaps. Therefore, the as typically practiced, initial discovery of splice junctions is more the purview of "mapping", rather than "aligning".

[0353] The mapping problem is that each read may be partitioned into exon segments at unknown boundaries, and the various exon segments are likely to map to widely separated genomic locations, which need to be individually discovered. Techniques to map exon segments to their corresponding reference locations can be similar to techniques to map a whole read to one reference segment, but spliced mapping (the former) is more challenging because each exon may be significantly shorter than the whole read, and therefore contains much less information to guide the mapper. Indeed, a single exon may be as short as one (1) base, such as "G", and without additional information it is not practical to determine where in the million-base potential intron range that single base should map to.

[0354] In addition to discovering the mappings of two consecutive exon segments of a read, the splice junction between them needs to be more precisely positioned, for at least some applications. Even though it may be clear that the first roughly 40 bases last roughly 60 bases of a 100-base read map to locations exactly 100,000 bases apart in Chromosome 3, it is often much less clear exactly how many read bases map to each of these two locations, or exactly where the splice junction, the boundary between the two exon segments, falls in the read. The correct CIGAR may plausibly be not just "40M100000N60M", for example, but "39M100000N61M" or "42M100000N58M". Hence, more precise positioning of splice junctions can be more of an "aligning" operation, rather than "mapping".

[0355] An RNA-capable mapper may also usefully infer which of the two DNA strands the read sequence was transcribed from. In typical non-directional RNA-seq protocols, a given read may align either forward or reverse-complemented to the reference (with or without splice junctions). In paired-end RNA-seq protocols, commonly the two mate reads are oriented "FR" (forward/reverse), such that the mate mapping earlier in the reference genome is oriented forward, and the other mate is reverse-complemented. But in typical non-directional RNA-seq protocols, these mapping orientations do not determine which DNA strand carried the gene from which the RNA for this read was transcribed, in part because both orientations are produced when cDNA is amplified by PCR.

[0356] Finally, an RNA-capable mapper can usefully leverage an input database of "annotated" known splice junctions. All common human genes have been studied in detail, for example, and the splice junctions of most common and less common RNA transcripts annotated in genomic databases. This information is not 100% comprehensive; any individual sample is likely to exhibit some "novel" splicing not recorded in databases. But still, annotated splice junctions can serve as useful guides to enhance the accuracy of RNA-seq mapping and alignment. After mapping/aligning RNA-seq reads with or without annotated splice junctions, another method may include the detection of the set of splice junctions observed in the aligned reads, with some criteria such as minimum number of alignments covering a splice junction, and this set of empirically detected splice junctions may be used as the annotated splice junctions for a second pass of RNA mapping/aligning. This can enhance sensitivity, by using splice junctions found in some reads to guide mapping of other reads.

[0357] Initial seed mapping for RNA-seq reads proceeds similarly to DNA reads. A primary seed length K is chosen, ideally somewhat longer than the base-4 logarithm of the reference genome size to make seeds map fairly uniquely, such as K=18 or K=21 for a whole human genome reference. A hash table is constructed, populated with some or all seeds from the reference genome, the hash record in the hash table for each populated seed indicating its position and orientation in the reference. The hash table is loaded into memory accessible to the mapper engine hardware, such as DRAM modules on an FPGA board wired to pins on the FPGA instantiating the mapper and/or other engine hardware.

[0358] The mapper engine receives RNA reads originating from an RNA or DNA sequencer (often having been reverse-transcribed into cDNA before sequencing). From each read, the mapper extracts seeds of length K, ideally a sliding window of multiple overlapping K-base seeds, chosen with some pattern, such as starting at each base position, or starting at every even-numbered position. The mapper accesses the hash table in memory for each seed, obtaining a list of zero, one, or more positions in the reference genome where the seed matches. As with DNA mapping, seeds may be dynamically extended, accessing the hash table repeatedly with successively longer seeds when necessary to reduce a large set of matching positions to a reasonably small set, such as 16 or fewer matches. Seed matches are aggregated into seed chains, comprising seeds matching with the same orientation (forward or reverse-complemented with respect to the reference) along similar alignment diagonals.

[0359] For RNA-seq reads, an additional step by the mapper engine to refine initial seed mapping with anchored short seed mapping may be advantageous. For instance, as can be seen with respect to FIG. **1**, RNA reads often cross one or more splice junctions, and a seed crossing a read's splice junction usually fails to map because its true image in the reference is split between two locations. When a read contains an exon shorter than the initial seed length K, or the read overlaps a longer exon by fewer bases than K, then seed mapping may fail to locate the corresponding reference position for that exon. Even when a whole or partial exon is somewhat longer than K bases, but shorter than the whole read, it can be vulnerable to seed mapping failure when it contains at least one edit (difference) from the reference, such as a single nucleotide polymorphism (SNP) or insertion or deletion (indel) from a mutation in the sample relative to the reference or from a sequencing error. For example, as can be seen with respect to FIG. **1**, an example of a failure to map all exon segments with long (K-base) seeds is shown. For this reason, for good seed mapping sensitivity, it is desirable to query shorter seeds, which can fit in short exons or short read-overhangs of exons, or between edits.

[0360] It may be somewhat impractical to query a whole-genome hash table for seeds much shorter than a minimum length related to the base-4 logarithm of the reference genome size, because shorter seeds will tend to match very large numbers of locations. For instance, with a whole human genome reference of size approximately 3.1 billion bases, the base 4 logarithm is approximately 15.8, and a minimum practical seed length to query may be K=16 or 18, with perhaps K=21 or more being an efficient setting; however, it is impractical to query K=11 base seeds, because each 11-base pattern will match an average of more than 700 reference locations.

[0361] However, after initial seed mapping with e.g. K=21 base seeds, it is possible to refine seed mapping with anchored seeds of a shorter length, such as L=11 bases. For anchored seed mapping, an anchored-seed hash table (which can be the same as the primary hash table, e.g., described above, or a separate one) is populated with L-base seeds from the reference, which are keyed to specific regions of the reference, such as bins of some size, such as 2^{16}=65,536 bases. Each reference region or bin is given a unique ID, such as its starting position in the reference genome divided by the bin size. L-base seeds within each reference bin are populated into the anchored-seed hash table, using a hash key formed from the L seed bases and the bin ID.

[0362] The mapper engine may query the anchored-seed hash table for any given L-base seed within any given bin, using a query hash key formed in the same manner from the L seed bases and the bin ID. Only L-base seed matches within that specific reference bin will be located by this query. Since the bin is much smaller than the whole reference genome, the short L-base seed has enough information to often map uniquely. For example, the base-4 logarithm of bin size 65,536 is 8, so L=11 (or 10, 12, etc.) is a practical anchored seed length to populate and query. As can be seen with respect to FIG. **2**, short (L-base) seeds more easily fit into short exons, short exon overhangs, or exon segments cut by edits such as SNPs.

[0363] A key to make anchored seed mapping work more efficiently is that mapper engine queries to the anchored-seed hash table may be guided by the results of initial seed mapping. Initial matches with, e.g., K=21, base seeds may not successfully map all exon segments of a read, but they are very likely to map at least one exon segment of each read, or of its paired end mate read. Given at least one K-base match within at least one exon segment in an RNA read or its mate, any other exon segments in the read which were not successfully mapped by K-base seeds are very likely to match relatively nearby in the reference genome.

[0364] For example, roughly 99% of human introns are shorter than 65,536 bases, so if one exon segment maps with K-base seeds to a given reference position, then other unmapped exon segments are likely to match within the same 65,536-base reference bin, or an adjacent bin. As can be seen with respect to FIG. **3**, a search range can be defined, e.g. the bin size, or ½ or ¼ the bin size, or twice the bin size, and one or more reference bins within the search range of successfully-mapped K-base seeds can be queried in the anchored-seed hash table using L-base seeds. Thus, K-base seed matches serve as anchors for local searches with shorter L-base seeds. This is likely to find additional matches to previously unmapped exon segments of the read. In this manner, seed mapping sensitivity is improved for RNA reads.

[0365] Additionally, there are various other ways that the mapper engine can utilize anchored short seed mapping. In one embodiment, after the mapper queries K-base initial seeds in the hash table and aggregates matches into seed chains, the mapper then extracts L-base seeds from the read, and queries these in nearby reference bins (within the selected search radius of current seed chains) to find additional matches to shorter L-base seeds, which the mapper engine then aggregates into additional seed chains, or adds to existing seed chains with similar alignment diagonals. In such an embodiment, it may be advantageous for the anchored-seed hash table to be the same as the primary hash table, or for distinct primary and anchored-seed hash tables to reside in accessible memory simultaneously. In either case, to fit the hash table(s) with both K-base and L-base seeds in memory, roughly twice as much memory may be used, such as 64 GB of DRAM rather than 32 GB of DRAM, or alternatively, roughly half as many reference seeds of each length may be populated, such as 50% populated seed density rather than 100% populated seed density. To limit the number of anchored-seed hash table queries required, only the more promising initial seed chains may be used as anchors, and/or L-base seeds may be extracted from the read only from certain regions, such as regions where K-base seeds did not successfully map.

[0366] In another embodiment, mapping and/or alignment for a set of reads may be taken to completion in a first pass using K-base seeds only. The mapping/alignment results for each read may then be examined, such as by software outside the mapper engine, e.g., on the CPU of an associated computer or server, to determine which reads require refined mapping using anchored short seeds. One indication that may trigger anchored seed refinement is that first-pass alignments are clipped, especially with clipping near or greater than the short seed length L. Another indication triggering anchored seed refinement may be a substantial amount of mismatching observed within the first-pass alignments. A further indication triggering anchored seed refinement may be that paired-end mates did not both map successfully, or mapped far away from each other or in unexpected relative orientations. Advantageously, if one read is selected for short seed refinement, its paired-end mate may also be selected. Advantageously, only a fraction of first-pass alignments may require short-seed refinement, such as about 15% to about 50%, including about 30%.

[0367] For each read in the subset employing short-seed refinement, one or more reference bins to search may be selected, such as bins overlapping a search radius around first-pass alignment results for the read and/or its mate (See FIG. **3**). Then a second mapping/alignment pass may be made over the subset of reads chosen for refinement. In the second pass, L-base seeds from each read may be queried in the anchored-seed hash table, keyed to the one or more selected reference bins for each read. Typically, for at least some of the reads selected for the second pass, improved mappings/alignments result, such as with higher alignment scores; the second-pass results can be retained in such cases when they are improved, or the first-pass results retained in other cases. Optionally, the primary hash table may be loaded into engine-accessible memory before the first mapping pass, and the anchored-seed hash table may be loaded before the second mapping pass, eliminating the need to fit both hash tables (or a single combined hash table) in memory at once, albeit both may be loaded at the same time, or to reduce reference seed population density to make both fit at once.

[0368] In some embodiments, the reference bins may have a configurable size, the search radius may be configurable, and both the initial seed length (K) and anchored seed length (L) may be configurable. In other embodiments, the reference bin size may be a power of two. Exemplary settings for human whole-transcriptome RNA-seq processing may be K=21, L=11, reference bin size 2^16=65,536, and search radius 2^14=16,384.

[0369] If annotated splice junctions are provided to the mapper engine, they can be leveraged to improve mapping sensitivity. The list of annotated junctions may be loaded into memory accessible by the mapper engine. Advantageously, the annotated junctions may be formatted into a table easily accessed by the mapper engine, such as a table with an entry for each e.g. 1024-base bin of the reference, which either contains information about an intron with at least one endpoint in that bin, or points to a list (in space allocated after the initial table) of multiple intron descriptors. Each intron descriptor indicates the reference positions of both endpoints of an associated intron, and may also carry additional information such as which DNA strand the intron's gene is on, the intron's motif, and a measure of how frequently the splice junction occurs.

[0370] After seed mapping (initial seeds and/or anchored short seeds) and seed chain formation, the annotated junction table may be accessed, at rows corresponding to the reference regions spanned by each seed chain, or regions near the ends of long seed chains. A list of introns with at least one endpoint nearby may be obtained, and may be compared with at least the seed chain for which the access was made. Each intron may be discarded if it is not a possible or likely splice junction from the seed chain. In particular, by comparing the intron endpoint location in the reference with the seed chain endpoint in the reference and in the read, an effective location of the splice junction in the read may be calculated. If this effective location is outside the bounds of the read, or overlaps the seed chain substantially (e.g. more than maxSpliceOlap=16 inside the seed chain's endpoint), or is too far outside the extents of the seed chain in the read (e.g. more than maxSpliceGap=150 bases outside the seed chain), then the annotated junction may be discarded as unlikely to be relevant.

[0371] Each remaining intron descriptor may then be considered as a possible splice junction from one end of the associated seed chain. This information may typically be utilized in two ways. First, the opposite end of the intron in the reference may be taken as a likely location that an adjacent portion of the read should map to, even if that location was not discovered by seed mapping. Indeed, the most likely alignment diagonal at the opposite end of the intron may be calculated by adding or subtracting (depending on orientations) the intron length from the alignment diagonal at the corresponding end of the current seed chain. If that reference location and alignment diagonal are not consistent with any existing seed chain, then a new (pseudo) seed chain may be fabricated starting at the reference location at the opposite end of the intron, and starting in the read at the corresponding position implied by the calculated alignment diagonal. In this manner, likely mapping locations of exon segments of the read may be discovered without seeds mapping inside of them, such as by inferring their locations across introns from existing seed chains.

[0372] Second, annotated intron information may be used to establish a known link between two seed chains, which represent adjacent exon segments in the read. Link information is recorded in one or both seed chain descriptors, identifying the other chain that it links to via an annotated splice junction. Furthermore, the position of the splice junction may be known (e.g., assuming the annotated junction is correct), calculated by differences between annotated intron endpoints and seed chain alignment diagonals. This splice junction positioning may also be recorded in one or both seed chain descriptors.

[0373] If multiple annotated splice junctions are discovered linking from the same seed chain, the link and splice junction position information can be recorded in various ways. For instance, each link between two chains may be recorded such as in only one of the two chains, so there may be no conflict if, for example, it is always recorded at the "destination" end of a link. One seed chain descriptor can have room to store multiple links, or have dynamic space for link information. Additionally, copies of existing seed chain descriptors can be made to hold alternate link information.

[0374] Annotated splice junction lookup may advantageously be iterated. Starting from one seed chain covering, for example, the first ⅓ of a read, an annotated splice junction may be discovered, linking to a previously undetected reference location, which may be fabricated into a new seed chain. The annotated junction table may be accessed again for the newly fabricated seed chain, perhaps discovering that after a second ⅓ of the read, there is another known junction to another undiscovered reference location. Advantageously, the annotated junction table entries can indicate the distance (continuing in the same direction as the junction annotated) before the nearest other annotated junction is reached, within transcripts of the same gene, or in general. When this distance, measured after the calculated splice junction location in the read, extends beyond the end of the read, there is no need to access the annotated junction table again, because nothing will be found.

[0375] Within the mapper engine, seed matches with same orientation (forward or reverse-complement with respect to the reference) and similar alignment diagonals are aggregated into seed chains, with the intent that a single gapless or gapped alignment operation may later examine and score the alignment between the read and the reference, such as for each seed chain. An alignment diagonal can be imagined as the diagonally-oriented alignment path covered by a matching seed, in the alignment rectangle, e.g., virtual matrix, formed with the read sequence on one axis and the reference sequence on the other axis; one representation as an integer may be calculated for forward alignments by subtracting a seed's position in the read from its position in the reference, and for reverse-complemented alignments by adding the seed's read position to its reference position.

[0376] When a read matches a segment of the reference exactly, such as positions 0 to 100 in the read matching positions 1,200,000 to 1,200,100 in the reference, all seeds normally match on the same diagonal, e.g. 1,200,000−0=1,200,100−100=1,200,000; a particular 21-base seed from bases 30 to 50 in the read would match bases 1,200,030 to 1,200,050 in the reference, also on the same diagonal 1,200,030−30=1,200,000. Seed matches with the same orientation and diagonal may be included in the same seed chain, but also seeds on slightly different alignment diagonals may be included in the same seed chain, such as seeds whose diagonals differ by no more than 20 or no more than 50, or some more complex rule. Allowing some tolerance for diagonal differences is useful because reads sometimes contain indels (insertions or deletions) with respect to the reference, and gapped alignment such as Smith-Waterman alignment in the aligner engine can resolve and score such indels for a single seed chain, as long as the indels are not too large, such as no more than 50 or 100 or 500 or 1,000 or more bases inserted or deleted.

[0377] But RNA-seq reads often cross splice junctions, at which a step from one read base to the next read base skips over a whole intron in the reference, which may be thousands of bases long, or even more than a million bases long. In such cases, seeds from one side of the splice junction in the read will map to the reference with dramatically different alignment diagonals from those on the other side of the splice junction; the diagonal-integer difference being equal to the length of the intron skipped, possibly thousands or more than a million. Such seeds are typically not be admitted to the same seed chain, because a gapped aligner may not be configured to directly resolve such a long gap in the reference.

[0378] So, for RNA mapping, unlike for DNA, it is to be expected that the true alignment of a given read may comprise multiple seed chains, each seed chain corresponding to a different exon segment in the read. Each candidate alignment, therefore, may include a sequence of several seed chains. A next stage in the mapper engine may be determining such candidate sequences of seed chains, known herein as scaffolds.

[0379] Each scaffold, as a sequence of one or more seed chains, has a physical interpretation as a piece-wise alignment of consecutive exon segments of the read to corresponding exon segments in the reference genome. As such, each seed chain in one scaffold should typically cover only a portion of the read, these portions progressing from the beginning of the read toward the end of the read along the sequence of seed chains; and the seed chains' corresponding reference segments should progress in a fixed direction through the reference, with intervening gaps corresponding to expected intron lengths. Each scaffold may be passed to the aligner engine, to resolve precise alignments and score them, and select the most likely candidate. But obtaining the list of scaffolds from the raw list of seed chains may be challenging.

[0380] In practice, a seed mapping for a single RNA-seq read may yield from a small number of seed chains to dozens or more than a hundred or a thousand seed chains. Given more than a hundred seed chains, the number of potential seed-chain sequences is astronomical. There is a problem, therefore, both of obtaining a reasonably short list of scaffolds for consideration in the aligner engine, and of determining that list of scaffolds from a given list of seed chains in a reasonable amount of time, so as not to slow down the mapper engine. A recursive method is herein presented for doing this efficiently.

[0381] First, it is useful to sort seed chains in order of their covered positions in the read, such as in increasing order of the seed chains' start positions in the read. Seed chains may naturally be constructed in such an order, by querying seeds in the hash table in order from the beginning of the read to the end, and forming them into seed chains in that same order. But if this is not the case, or if the order is disturbed by subsequently modifying the list of seed chains with anchored-seed mapping or lookups of annotated splice junctions, then the seed chains may be sorted before scaffolding, such as using a "quicksort" or other sorting algorithm.

[0382] Next, rules may be established under which one seed chain (B) may be allowed to immediately follow another seed chain (A) in the same scaffold, establishing a seed-chain link from A to B. There is considerably flexibility in rules that can work well, but the rules should permit likely seed-chain links in a true-alignment scaffold, while excluding as many unlikely seed-chain links as possible. Here are various sets of rules, with various named parameters and good default values, that may be employed.

[0383] Criteria for seed chain B to follow seed chain A in a scaffold:

[0384] A & B have same orientation

[0385] (Gap between A & B in the read)=:gap≤maxSpliceGap=150

[0386] (Overlap between A & B in the read)=:olap≤maxSpliceOlap=16

[0387] (Gap between A start and B start in the read)=:head≥olap+(olapAdj=4)

[0388] (Gap between A end and B end in the read)=:tail≥olap+(olapAdj=4)

[0389] A/B reference gap minus A/B read gap)=:intronLen≥minIntronLen=20

[0390] (A/B reference gap minus A/B read gap)=:intronLen≤maxIntronLen=1,000,000

[0391] When annotated splice junctions are used, and an annotated link has been recorded between seed chains A and B, then they may be allowed to follow each other.

[0392] Here is a recursive algorithm to form multiple scaffolds:

[0393] Sort N seed chains by start position in the read, if necessary

[0394] Loop c0=0 to N−1

[0395] Skip c0 if already used inside any scaffold

[0396] Initialize last=0,scaf[0]=c0,start=c0+1,stack [0]=1, stack[1]=0,stackPos=0

[0397] Loop while last ≥0

[0398] Loop c=start to N−1

[0399] If chain c can follow chan scaf[last]:

[0400] scaf[++last]=c

[0401] stack[last]=0 if stack[last]=c

[0402] Else if last>stackPos and chain c can follow chain scaf[last−1]:

[0403] stack[last]=c

[0404] Output scaffold scaf[0. . . last]

[0405] Set stackPos=maxiumum in (0. . . last) with stack [stackPos]>0

[0406] Set start=scaf[stackPos]+1

[0407] Set last=stackPos−1

[0408] Term and variable meanings in the algorithm:

[0409] "chain":index 0. . . N−1 of a seed chain

[0410] scaf[]=scaffold under construction, each slot getting a chain 0. . . N−1

[0411] c0=first chain in scaffold (slot 0)

[0412] last=end slot ♯(so far) in scaffold

[0413] start=first chain in search loop

[0414] stack[i]=highest-numbered alternative chain for scaf[i], or 0 if none. This represents the endpoint of the search for alternatives for scaf[i]after recursion backup.

[0415] stackPos=the target scaffold slot to replace via recursion backup. Observe that when the backup occurs stack[stackPos] is baked in, and will not be updated unitil it is cleared.

[0416] This recursive search may be implemented in physical logic within the mapper engine. There can be time available to execute this algorithm without significantly slowing down the engine, using methods of hardware parallelism. Specifically, a batch of seed chains for a given read can be buffered for scaffolding logic to process downstream in a processing pipeline, in parallel with seed mapping and chaining logic processing the next read.

[0417] Recursion may occasionally get carried away, so it may be useful to limit it. A useful way to limit recursion while also limiting the set of scaffolds produced is to filter inferior scaffolds as they are produced. A useful scaffold filtering method, therefore, is presented herein. For each scaffold, its net coverage of the read may be calculated, such as a count of read bases covered by one or more seed chains in the scaffold. Higher coverage scaffolds are more likely to represent the true alignment. In particular, if the maximum read-coverage among all scaffolds formed (so far) is tracked, then scaffolds with a large coverage delta behind the maximum coverage are less likely to represent the true alignment.

[0418] Also, for each scaffold, its net span in the reference genome can be calculated, e.g., the distance between the outermost bases in the reference of the first and last seed chains in the scaffold. Scaffolds with very large reference spans are less likely to represent the true alignment. Combining these two measurements as follows is especially powerful for scaffold filtering: filter_metric=(max_coverage−coverage)+floor(25*($\log_2$(ref_span+2^{13})−13). The constants 25 and 13 should be configurable parameters: rna-filt-ratio=25, and rna-span-log-min=13. Filter out all multi-chain scaffolds where this metric exceeds a configurable threshold, rna-max-covg-gap=150 for example. A threshold of 200 makes the filter considerably looser, and 100 considerably tighter.

[0419] This filter can be applied to a complete or incomplete set of finished scaffolds produced from the list of seed chains for a given read, by tracking or calculating the maximum coverage among all the scaffolds, and scanning the list of scaffolds, discarding those with filter_metric>rna-max-covg-gap.

[0420] Such a filter can also be applied as recursion pruning in the middle of scaffold formation. As each new seed chain is added to a scaffold, an updated reference span may be noted, and also a potential coverage, calculated by subtracting coverage gaps within and preceding this seed chain from the read length. Using this partial span and potential coverage, if the metric would trigger filtering, then any longer scaffold using the current partial scaffold as a prefix may be filtered, because reference span will only increase and potential coverage will only decrease. Therefore, in such instances, all recursion retaining the current partial scaffold as a prefix can be skipped. Pruning recursion with the scaffold filter in this manner can significantly reduce the length of recursion to form a set of scaffolds from a long list of seed chains.

[0421] Performance of the recursive scaffold search can also be optimized. The recursive loops repeatedly scan the portions of the list of seed chains, and speed of the algorithm is therefore much better when the list of seed chains is shorter. But it is not actually necessary to execute the recursive algorithm on the entire list of seed chains, when some seed chains cannot possibly scaffold with other seed chains. One way to optimize is to detect "isolated" seed chains, which are located farther than maxIntronLen (e.g. 1,000,000 bases) from any other seed chain in the reference. Isolated seed chains can be emitted automatically as single-chain scaffolds, and removed from the list of seed chains before further scaffolding, thus shortening the list of seed chains scanned during recursion. Likewise, well-separated subsets of seed chains could be detected, such as subsets within each chromosome, or subsets separated by more than maxIntronLen in the reference, and the recursive scaffolding algorithm can be executed separately on each such subset, resulting in significantly reduced total execution time.

[0422] A first aligner engine step for each scaffold may be to position each splice junction, the boundary between two exon segments (represented by two corresponding seed chains in the scaffold). This is called "stitching" the exon segments together, or stitching the splice junctions. More precise stitching may be needed after seed mapping and scaffolding, because two successive seed chains by themselves may not make clear where the true boundary between them lies. For instance, successive seed chains may be separated by some distance in the read, if seeds were unable to map immediately on one or both sides of the splice junction; or successive seed chains may overlap each other in the read, especially if the read sequence at the end of one exon matches the sequence at the beginning of the next exon. Even if successive seed chains abut with no gap or overlap, it is not guaranteed that the boundary between them lies at the true position of the splice junction.

[0423] Splice junction stitching is thus an analysis to select the best stitching position between successive exon segments in the read, corresponding to the most likely splice junction position. Two factors are useful in determining this. The first factor is comparison of the read sequence with the left and right reference sequences, at the two exon-segment mapping locations in the reference genome. A given stitching position implies that read bases left of the stitch map to the left reference region, and reads bases right of the stitch map to the right reference region. As a potential stitch position is moved from left to right in the read, read bases switch their mapping as they are crossed, from the right reference region to the left one.

[0424] As can be seen with respect to FIG. **4**, the true splice junction position is likely to have good matching between the leftward portion of the read and the left reference region, and between the rightward portion of the read and the right reference region. The total number of mismatches (or SNPs) can be counted on both sides of a potential stitch position, by comparison with the corresponding reference region; and stitch positions with smaller SNP counts are more likely to be true. The comparison of read portions left and right of each stitch position is illustrated in FIG. **4**.

[0425] This SNP counting is modeled efficiently by scanning stitch positions through a window of possible positions in the read, such as overlapping each of the two seed chains at most some distance, e.g., such as 48 bases. This scanning runs in the hardware aligner engine, for example, at a speed of one position per clock cycle. Each time the scan moves one step, such as from left to right, only one read base switches its mapping, from the right reference region to the left reference region. Therefore, the left sequence comparison either gains one SNP or remains the same, and the right sequence comparison either loses one SNP or remains the same; and thus the net SNP count changes by −1, 0, or +1. This incremental SNP count change for each step can be calculated by comparing one read base (the one crossed by the stitch position step) with two reference bases. If this incremental SNP count change is summed as steps are taken from left to right, then the current sum can be taken as a relative score, where the minimum score is best. Equivalently, each matching base can be given a positive match score, and each mismatching base a negative mismatching penalty; and the sum of incremental score changes should be maximized for the best stitch position.

[0426] Another factor is the intron motif implied by each stitch position. The intron motif is defined as the first two bases and last two bases of the skipped reference segment, or intron. Equivalently, the motif for any potential stitch position is formed from the first two bases after the left reference region and the last two bases before the right reference region, as shown in FIG. **4**. This implied intron motif is dependent on the stitch position, and usually varies as the stitch position scans across the window of possible stitch positions. Certain “canonical” intron motifs occur much more commonly than others in natural RNA splicing. A stitch position that corresponds to a canonical intron motif is more likely to be the true splice junction position, especially if it is one of the more common canonical motifs.

[0427] A table of three canonical intron motifs in human RNA is shown in TABLE I, below. For each motif, its reverse-complement is also shown, because in most RNA-seq protocols the reads may map either forward or reverse-complemented with respect to the transcribed gene strand, so although only the “forward” canonical motif occurs in the original transcribed gene strand, its reverse-complement can appear in RNA-seq reads. For each canonical motif, and for the remaining class of non-canonical motifs, an approximate frequency in human RNA splicing is shown, along with a sample score penalty, which may be used, for example, with base matches scoring +1 and base mismatches scoring −4.

TABLE I

Intron Motif	Reverse-Complement	Approx. Frequency	Score Penalty
GT/AG	CT/AC	98.73%	0
GC/AG	CT/GC	1.03%	10
AT/AC	GT/AT	0.11%	15
250 non-canonical motifs		0.13%	25

[0428] The splice stitching module accordingly scans a potential stitch position across a window of possible stitch locations, such as from left to right, summing incremental score changes due to bases switching which reference region they map to, and also subtracting at each potential stitch position an intron motif penalty according to the intron motif observed just after the left reference region and just before the right reference region, and chooses the maximum scoring position to stitch.

[0429] Additionally, certain special outcomes may be considered and scored. Stitching at the left edge or right edge of the window of possible stitch positions may be considered failure to stitch, and is likely to arise when one of the two reference regions is not a true mapping position for a read exon segment, such as when an annotated splice junction was followed, but turns out not to be true for this read. Left or right edge stitching can advantageously be given a scoring bonus, such as 25 in the same exemplary scoring scale, so that significant evidence of a true splice junction must appear for stitching to succeed.

[0430] Also, if an annotated splice junction was identified linking the two seed chains being stitched, the annotated junction is at a known position within the window of possible stitch positions. As one option, the stitching operation can be skipped, simply accepting the annotated junction’s known position. As another option, the stitching operation can be performed, but the known position of the annotated junction may be given a score bonus, and/or may automatically be given the best available intron motif penalty, or a zero penalty. As another option, in lieu of an intron motif penalty, the known position of the annotated junction may be given a score bonus or penalty associated with the observed commonality or rarity of that splice junction as noted in annotation databases. If the annotated splice junction’s known position is selected for stitching, then the stitched junction may be flagged as in agreement with an annotated junction, so this fact can be reported if this splice junction appears in the read’s output alignment.

[0431] Selected stitch positions can be annotated into scaffolds in various manners. In a preferred embodiment, the constituent seed chains of a scaffold are edited to begin and end immediately adjacent to selected stitch positions.

[0432] Additionally, it is advantageous for the aligner engine to make larger scaffold edits in some circumstances, based on stitching results. If stitching fails, then the scaffold may be truncated, or split into two scaffolds at the failure point. Also, stitching may be attempted between non-adjacent seed chains in the scaffold, such as skipping a single seed chain. For instance, for a scaffold containing seed chains 1, 2, 3, and 4, splice junction stitching should naturally be performed between the chains pairs (1,2), (2,3), and (3,4); but in addition, stitching may be attempted between chain pairs (1,3) and (2,4). If stitching from 1 to 3 scores better than stitching from 1 to 2 followed by 2 to 3, then seed chain 2 may be dropped from the scaffold.

[0433] Having determined precise splice junction positions in candidate scaffolds by stitching, corresponding complete alignments and alignment scores can be determined for each scaffold by use of a gapless aligner or gapped aligner (such as Smith-Waterman) module, as herein described. For a scaffold with only a single seed chain, this is not significantly different than alignment for DNA reads, and the same hardware modules and methods, as herein described, can be used. For a scaffold with multiple seed chains, some further method may be needed to obtain a complete, possibly spliced (containing intron operations) alignment.

[0434] One method by which complete spliced alignments can be determined is to separately align each exon segment the read, corresponding to each seed chain in the scaffold, to its corresponding reference segment, with a gapless and/or gapped aligner. This has disadvantages in when local (e.g., possibly clipped) alignments are desired. If individual exon segment alignments are produced without clipping, then they are not easily assembled together into a complete spliced alignment. If individual exon segment alignments are produced without clipping, then their alignment scores can be examined to determine if the best overall local alignment would clip off one or more entire exon segments, but appropriate clipping at arbitrary locations within the exon segments is not easily determined. It may therefore expensive to produce both clipped and unclipped versions of each exon segment alignment to resolve these difficulties.

[0435] One method for determining complete spliced alignments for a multi-chain scaffold involves concatenating exon segments together before aligning. Each aligner module—gapless or gapped—may be fed two nucleotide sequences to align, a query (read) sequence and a reference sequence. The concatenated query sequence may be simply the entire read, which is the concatenation of its exon segments, with optional clipping of the beginning or end of the read if the first or last exon segment does not extend to the read beginning or end.

[0436] The concatenated reference sequence is obtained by fetching the reference genome segment that is the mapped image of each exon-segment seed chain, and concatenating these reference segments together. Note that for a given exon segment (seed chain), its reference segment may be a different length than its segment of the read, in a case where the leftmost seeds in the seed chain fell on a somewhat different alignment diagonal than the rightmost seeds; e.g., the seeds in the seed chain imply the presence of an indel. In such a case, a gapped aligner should be used.

[0437] Furthermore, for gapped alignment, the first and last exon segments of the reference sequence may be extended outward, for example extending the first exon segment with 50 preceding reference bases and the last exon segment with 50 following reference bases, to provide room for deletions within the first and last exon segments.

[0438] By concatenating the scaffold's exon segments from the read and reference, a single concatenated query sequence and a single concatenated reference sequence can be fed to the aligner module, which can therefore function in much the same manner for spliced RNA alignments as for unspliced (RNA and/or DNA) alignments. However, some further modifications may be advantageous. First, to determine appropriate clipping of local alignments at any position within any exon segment, score penalties (or bonuses) may be applied at each splice junction the alignment crosses. In a preferred embodiment, a score penalty for each splice junction is related to its intron motif and annotated splice junction status, and may be the same score penalty used in splice junction stitching.

[0439] Accordingly, an unannotated splice junction with rare or non-canonical intron motif may have a large associated score penalty, and one or more whole exon segments become more likely to get clipped from the spliced alignment in order to exclude such an unlikely splice junction, unless enough sequence matching occurs on both sides of the splice junction to serve as convincing evidence the splice junction is really present by overcoming its score penalty. In a preferred embodiment, the concatenated query and reference sequences each have a dummy base inserted between successive exon segments, and the appropriate score penalty for each splice junction is attached to its corresponding dummy base. This allows the splice junction score penalties to be included without specialized logic, and provides room for possible alignment clipping on either side of the splice junction dummy base.

[0440] Additionally, in various instances, for gapped alignment such as Smith-Waterman, it can be advantageous to force alignment paths to pass through the predetermined splice junctions. In other words, no alignment path should cross from one query exon segment to the next without simultaneously crossing from the corresponding reference exon segment to the next. One reason for this restriction is that only the properly synchronized splice junction will score properly, based on the intron motif determined during splice junction stitching.

[0441] Another reason is that the concatenated reference sequence has been formed using the precise reference exon segment boundaries corresponding to the selected stitch positions of each splice junction, so there are not additional reference bases for the gapped aligner to adjust splice junctions freely. Furthermore, to avoid difficult-to-interpret alignments (such as CIGAR strings with 'I' or 'D' operations adjacent to 'N' operations), it is desirable to require at least one query and reference base before each included splice junction to be "diagonally" aligned (query base aligned to reference base, as in a CIGAR 'M' operation), and at least one query and reference base after each included splice junction to be "diagonally" aligned. To enforce these restrictions, the concatenated reference and query sequences are divided into zones, which are assigned identifiers or zone IDs, such as integer values. In one embodiment, one zone ID is assigned to each dummy base between exon segments, another zone ID to the last base of each exon segment preceding a splice junction (but not the final base of the concatenated sequence), and another zone ID to all the remaining bases of each exon segment.

[0442] For instance, for a scaffold with three exon segments (seed chains) each 20 bases long, there could be 4 zone IDs: zone 1 for bases 1-19 of the first exon segment, zone 2 for base 20 of the first exon segment, zone 3 for the dummy base between the first and second exon segments, zone 4 for bases 1-19 of the second exon segment, zone 5 for base 20 of the second exon segment, zone 6 for the dummy base between the second and third exon segments, and zone 7 for bases 1-20 of the third exon segment. The same zone mapping applies to the both the concatenated query sequence and the concatenated reference sequence, bearing in mind that corresponding multi-base query and reference zones with the same zone ID may have different lengths due to indels in the seed chains. Then, in the gapless aligner, a scoring cell may be modified to allow a valid alignment score at the intersection between identical zone IDs, e.g., where the query zone ID matches the reference zone ID.

[0443] FIG. **5** shows the abstract alignment rectangle, with concatenated query sequence on the vertical axis and concatenated reference sequence on the horizontal axis. Dummy bases of each concatenated sequence are shaded (zones 3 and 6). A grid overlays the alignment rectangle to show the boundaries between zones on each axis. Sub-rectangles with matching zone ID are valid alignment regions, and other (shaded) sub-rectangles are invalid alignment regions. A valid exemplary alignment is shown, which is end-to-end in the query sequence, and contains an insertion (vertical segment) in the second exon segment (zone 4), and a deletion (horizontal segment) in the third exon segment (zone 7). The valid alignment passes diagonally through the splice junctions (zones 3 and 6).

[0444] Gapless or gapped alignment using concatenated query and reference sequences produces a correct alignment score, but the alignment trace (e.g. CIGAR string) requires editing, because it does not yet contain intron ('N') operations. For example, a spliced alignment of a 100-base read without indels may emerge from alignment with CIGAR "101M", meaning 101 bases aligned diagonally without indels. There are two adjustments needed in this CIGAR. First, the dummy base between exon segments is counted in the CIGAR, and should not be. Second, the intron operation, e.g. 895 bases long, needs to be inserted at the position of the dummy base. The correct CIGAR may be "40M895N60M", for example.

[0445] Given the scaffold with comprising seed chains defining the endpoints of the exon segments as stitched, it is straightforward arithmetic to locate the position of each splice junction in the CIGAR, remove "1M" for the dummy base, and replace it with an intron operation of the proper length. For local alignments, this process must account for the possibility that one or more whole exon segments were clipped from the alignment. The same process of arithmetic can calculate the correct start and end positions of the alignment in the reference genome.

[0446] Having obtained alignment scores, start and end positions, and CIGAR strings for each aligned scaffold, processing to select and output the best possibly-spliced alignment is similar to DNA processing. Paired end alignment candidates are examined to find properly positioned and oriented alignment pairs. Alignment candidate pairs, including non-properly-paired candidates, are given score penalties for being unpaired or having improbably empirical insert lengths; pair scores are formed by combining (such as adding) alignment scores from each mate and a pairing penalty; and the best scoring pair of alignments is chosen and output from the aligner engine.

[0447] Apparent insert length, usually measured as the span in the reference covered by the two mate read alignments, can appear extremely long due to introns within either or both mates, or unobserved introns in the gap between the mates. (Physical inserts are potentially much shorter, being the lengths of the sequenced RNA or cDNA molecule, where the introns are spliced out.) Therefore, much longer apparent insert lengths must be considered properly paired and given zero or small pairing penalties; this can be done according to a known intron length distribution in the sampled species, and/or the observed apparent insert distribution in the RNA-seq data being processed.

[0448] In a preferred embodiment, for each read processed, the alignment score, start position, and encoded CIGAR string are output from the aligner module. In addition, in some embodiments, for each splice junction in the alignment, its intron motif and annotation status are output. A mapping quality or confidence, such as a phred-scale "MAPQ" parameter, may also be estimated and output. In various embodiments, MAPQ is estimated primarily in proportion to the difference between the best pair score and the second-best pair score with a different alignment for the current read. Additional alignment candidates, or secondary alignments, may also be output for each read, such as a limited number of other candidates scoring within a defined or configured score difference threshold.

[0449] It is to be understood, such as with reference to the above, that although a mapping function may in some instances have been described, such as with reference to a mapper, and/or an alignment function may have in some instances been described, such as with reference to an aligner, these different functions may be performed sequentially by the same architecture, and although the above has been described with reference to RNA mapping and aligning, various of this methods and the devices for performing the same may be employed, where it makes sense to do so, in relation to the analysis of DNA. Accordingly, in various instances, both the mapping function and the aligning function, as herein described may be performed by a common architecture that may be understood to be an aligner, especially in those instances wherein to perform an alignment function, a mapping function need first be performed.

[0450] The output from the alignment module is a SAM (Text) or BAM (e.g., binary version of a SAM) file along with a mapping quality score (MAPQ), which quality score reflects the confidence that the predicted and aligned location of the read to the reference is actually where the read is derived. Accordingly, once it has been determined where each read is mapped, and further determined where each read is aligned, e.g., each relevant read has been given a position and a quality score reflecting the probability that the position is the correct alignment, such that the nucleotide sequence for the subject's DNA is known as well as how the subject's DNA differs from that of the reference (e.g., the CIGAR string has been determined), then the various reads representing the genomic nucleic acid sequence of the subject may be sorted by chromosome location, so that the exact location of the read on the chromosomes may be determined. Consequently, in some aspects, the present disclosure is directed to a sorting function, such as may be performed by a sorting module, which sorting module may be part of a pipeline of modules, such as a pipeline that is directed at taking raw sequence read data, such as form a genomic sample form an individual, and mapping and/or aligning that data, which data may then be sorted.

[0451] More particularly, once the reads have been assigned a position, such as relative to the reference genome, which may include identifying to which chromosome the read belongs and/or its offset from the beginning of that chromosome, the reads may be sorted by position. Sorting may be useful, such as in downstream analyses, whereby all of the reads that overlap a given position in the genome may be formed into a pile up so as to be adjacent to one another, such as after being processed through the sorting module, whereby it can be readily determined if the majority of the reads agree with the reference value or not. Hence, where the majority of reads do not agree with the reference value a variant call can be flagged. Sorting, therefore, may involve one or more of sorting the reads that align to the relatively same position, such as the same chromosome position, so as to produce a pileup, such that all the reads that cover the same location are physically grouped together; and may further involve analyzing the reads of the pileup to determine where the reads may indicate an actual variant in the genome, as compared to the reference genome, which variant may be distinguishable, such as by the consensus of the pileup, from an error, such as a machine read error or error an error in the sequencing methods which may be exhibited by a small minority of the reads.

[0452] Once the data has been obtained there are one or more other modules that may be run so as to clean up the data. For instance, one module that may be included, for example, in a sequence analysis pipeline, such as for determining the genomic sequence of an individual, may be a local realignment module. For example, it is often difficult to determine insertions and deletions that occur at the end of the read. This is because the Smith-Waterman or equivalent alignment process lacks enough context beyond the indel to allow the scoring to detect its presence. Consequently, the actual indel may be reported as one or more SNPs. In such an instance, the accuracy of the predicted location for any given read may be enhanced by performing a local realignment on the mapped and/or aligned and/or sorted read data.

[0453] In such instances, pileups may be used to help clarify the proper alignment, such as where a position in question is at the end of any given read, that same position is likely to be at the middle of some other read in the pileup. Accordingly, in performing a local realignment the various reads in a pileup may be analyzed so as to determine if some of the reads in the pile up indicate that there was an insertion or a deletion at a given position where an other read does not include the indel, or rather includes a substitution, at that position, then the indel may be inserted, such as into the reference, where it is not present, and the reads in the local pileup that overlap that region may be realigned to see if collectively a better score is achieved then when the insertion and/or deletion was not there. Accordingly, if there is an improvement, the whole set of reads in the pileup may be reviewed and if the score of the overall set has improved then it is clear to make the call that there really was an indel at that position. In a manner such as this, the fact that there is not enough context to more accurately align a read at the end of a chromosome, for any individual read, may be compensated for. Hence, when performing a local realignment, one or more pileups where one or more indels may be positioned are examined, and it is determined if by adding an indel at any given position the overall alignment score may be enhanced.

[0454] Another module that may be included, for example, in a sequence analysis pipeline, such as for determining the genomic sequence of an individual, may be a duplicate marking module. For instance, a duplicate marking function may be performed so as to compensate for chemistry errors that may occur during the sequencing phase. For example, as described above, during some sequencing procedures nucleic acid sequences are attached to beads and built up from there using labeled nucleotide bases. Ideally there will be only one read per bead. However, sometimes multiple reads become attached to a single bead and this results in an excessive number of copies of the attached read. This phenomenon is known as read duplication.

[0455] Such read duplication may throw off the statistics and create a statistical bias because instead of having an equal representation of all reads, various reads have been duplicated, such as because of the duplicate template sequences attached to more than one bead are over represented. Accordingly, these may be determined because any read that aligns to the exact same position, and has the exact same length, is likely a duplicate. Once this is identified by the system, only one read need be subjected to further processing and the others may be marked as duplicates and, therefore, can be discarded or ignored. A typical situation where this occurs is where there is not enough genetic material to process from the very beginning and the system attempts to overcompensate for that.

[0456] Another module that may be included, for example, in a sequence analysis pipeline, such as for determining the genomic sequence of an individual, may be a base quality score recalibrater. For instance, every base of every read has a Phred score that indicates the probability that the called base at that position is incorrect. For example, the Phred score for any base is due in part to the nature of the base that precedes it and the error profile will be different depending on which base precedes the base in question. Further, there is a greater likelihood of an error occurring at the ends of a read, e.g., such as where at the ends of the reads the chemistry is starting to lose its performance. A base quality score recalibration is a covariant analysis that may go back and measures the empirical quality of the base quality score as a function of all those things by which it varies.

[0457] In various instances, it involves two passes, the first gathers all the actual, empirical measured data and statistics on the error rate observed as a function of all the variables, and the second pass involves the actual recalibration of the scores by flowing all the reads through a filter modifying the quality scores for every single base as a function of the variables based on what was actually empirically measured in the data set. This compensates for all the differences in the data due to the various variables and cleans up that data and score. The purpose of all this cleanup is to ensure the best possible variant calling is achieved. Many variant callers base their decisions in part on the reported quality of each of the nucleotides that pile up at each position in the genome. If the quality scores are not accurate, there could easily result a wrong call.

[0458] Another module that may be included, for example, in a sequence analysis pipeline, such as for determining the genomic sequence of an individual, may be a compression module, which executes a compression function. As indicated above, it may be useful at some point to take the generated and processed data and transmit it to a remote location, such as the cloud, and hence, the data may need to be compressed at a particular stage of processing, whereby once compressed it may be transmitted and/or otherwise uploaded, such as on to the cloud or to a server farm, etc., for instance, for the performance of the variant calling module. The results once obtained may then be decompressed and/or stored in the memory, on a data base on the cloud, such as an electronic health and/or research database, and the like, which in turn, can be made available for tertiary processing, etc.

[0459] Specifically, as described above, the likelihood of a given read being associated with a given haplotype may be calculated for all read/haplotype combinations. In such instances, the likelihood may be calculated using a Hidden Markov Model (HMM).

[0460] For instance, the various assembled haplotypes may be aligned in accordance with a dynamic programming model similar to a SW alignment. In such an instance, a virtual matrix may be generated such as where the haplotype may be positioned on one axis of a virtual array, and the read may be positioned on the other axis. The matrix may then be filled out with the scores generated by traversing the extracted paths through the graph and calculating the probabilities that any given path is the true path. Hence, in such an instance, a difference in this alignment protocol from a typical SW alignment protocol is that with respect to finding the most likely path through the array, a maximum likelihood calculation is used, such as a calculation performed by an HMM model that is configured to provide the total probability for alignment of the reads to the haplotype, which HMM model may be implemented in software or hardware or a combination of both. Hence, an actual CIGAR strand alignment, in this instance, need not be produced. Rather all possible alignments are considered and their possibilities are summed. The pair HMM evaluation is resource and time intensive, and thus, implementing its operations within a hardwired configuration within an integrated circuit is very advantageous.

[0461] For example, each read may be tested against each candidate haplotype, so as to estimate a probability of observing the read assuming the haplotype is the true representative of the original DNA sampled. In various instances, this calculation may be performed by evaluating a “pair hidden Markov model” (HMM), which may be configured to model the various possible ways the haplotype candidate might have been modified, such as by PCR or sequencing errors, and the like, and a variation introduced into the read observed. In such instances, the HMM evaluation, as depicted in FIG. **6**, may employ a dynamic programming method to calculate the total probability of any series of Markov state transitions arriving at the observed read in view of the possibility that any divergence in the read may be the result of an error model. Accordingly, such HMM calculations may be configured to analyze all the possible SNPs and Indels that could have been introduced into one or more of the reads, such as by amplification and/or sequencing artifacts.

[0462] Particularly, PCR introduced errors can be modeled and accounted for based on the probabilities that such errors would occur. For instance, insertion and deletion base qualities can be calculated at each position, such as based on the type of errors that typically occur due to this process and the artifacts, e.g., tandem repeats, it routinely produces in the sequences it generates, which information may be inserted into the array, and in view of such respective base qualities may be adjusted. In such instances, the HMM process may generate the probability of all the multiplicity of all conceivable errors that could in combination produce the same read result hypothesis, because there are very many ways, e.g., modifications that can take place and still get to the same answer.

[0463] More particularly, paired HMM considers in a virtual matrix as presented in FIG. **7** all the possible alignments of the read to the reference haplotype along with a probability associated with each of them, where all probabilities are added up. The sum of all of the probabilities of all the variants along a given path through the matrix is added up to get one overarching probability for each read. This process is then performed for every pair, for every haplotype, read pair. For example, if there is a six pile up cluster overlapping a given region, e.g., a region of six haplotype candidates, and if the pile up includes about one hundred reads, 600 HMM operations will then need to be performed. More particularly, if there are 6 haplotypes then there are going to be 6 branches through the path and the probability that each one is the correct pathway that matches the subject's actual genetic code for that region must be calculated. Consequently, each pathway for all of the reads must be considered, and the probability for each read that you would arrive at this given haplotype is to be calculated.

[0464] The pair Hidden Markov Model is an approximate model for how a true haplotype in the sampled DNA may transform into a possible different detected read. It has been observed that these types of transformations are a combination of SNPs and indels that have been introduced into the genetic sample set by the PCR process, by one or more of the other sample preparation steps, and/or by an error caused by the sequencing process, and the like. As can be seen with respect to FIG. **6**, to account for these types of errors, an underlying 3-state base model may be employed, such as where: {M=alignment match, I=insertion, D=deletion}, further where any transition is possible except I<->D.

[0465] As can be seen with respect to FIG. **6**, the 3-state base model transitions are not in a time sequence, but rather are in a sequence of progression through the candidate haplotype and read sequences, beginning at position 0 in each sequence, where the first base is position 1. A transition to M implies position +1 in both sequences; a transition to I implies position +1 in the read sequence only; and a transition to D implies position +1 in the haplotype sequence only. The same 3-state model may be configured to underlie the Smith-Waterman and/or Needleman-Wunsch alignments, as herein described, as well. Accordingly, such a 3-state model, as set forth herein, may be employed in a SW and/or NW process thereby allowing for affine gap (indel) scoring, in which gap opening (entering the I or D state) is assumed to be less likely than gap extension (remaining in the I or D state). Hence, in this instance, the pair HMM can be seen as alignment, and a CIGAR string may be produced to encode a sequence of the various state transitions.

[0466] In various instances, the 3-state base model may be complicated by allowing the transition probabilities to vary by position. For instance, the probabilities of all M transitions may be multiplied by the prior probabilities of observing the next read base given its base quality score, and the corresponding next haplotype base. In such an instance, the base quality scores may translate to a probability of a sequencing SNP error. When the two bases match, the prior probability is taken as one minus this error probability, and when they mismatch, it is taken as the error probability divided by 3, since there are 3 possible SNP results.

[0467] In such instances, the 3 states are no longer a true Markov model, both because transition probabilities from a given state do not sum to 1, and because the dependence on sequence position, which implies a dependence on previous state transitions, and thus violates the Markov property of dependence only on the current state. Such a Markov property can be salvaged if one instead considers the Markov model to have 3(N+1)(M+1) states, where N and M are the haplotype and read lengths, and there are distinct M, I, and D states for each haplotype/read coordinate. Further, the sum of probabilities to 1 can be salvaged if an additional "FAIL" state is assumed, with transition probability from each other state of (1−MPriorProb)(MTransProb). Furthermore, the relative balance of M transitions vs. I and D transitions also varies by position in the read. This is according to an assumed PCR error model, in which PCR indel errors are more likely in tandem repeat regions. Thus, there is a preprocessing of the read sequence, examining repetitive material surrounding each base, and deriving a local probability for M->I and M->D transitions; M->M transitions get the remainder (one minus the sum of these two), times the M prior.

[0468] The above discussion is regarding an abstract "Markovish" model. In various instances, the maximum-likelihood transition sequence may also be determined, which is termed herein as an alignment, and may be performed using a Needleman-Wunsch or other dynamic programming algorithm. But, in various instances, in performing a variant calling function, as disclosed herein, the maximum likelihood alignment, or any particular alignment, need not be a primary concern. Rather, the total probability may be computed, for instance, by computing the total probability of observing the read given the haplotype, which is the sum of the probabilities of all possible transition paths through the graph, from read position zero at any haplotype position, to the read end position, at any haplotype position, each component path probability being simply the product of the various constituent transition probabilities.

[0469] Finding the sum of pathway probabilities may also be performed by employing a virtual array and using a dynamic programming algorithm, as described above, such that in each cell of a (0 . . . N)×(0 . . . M) matrix, there are three probability values calculated, corresponding to M, D, and I transition states. (Or equivalently, there are 3 matrices.) The top row (read position zero) of the matrix may be initialized to probability 1.0 in the D states, and 0.0 in the I and M states; and the rest of the left column (haplotype position zero) may be initialized to all zeros. (In software, the initial D probabilities may be set near the double-precision max value, e.g. 2^1020, so as to avoid underflow, but this factor may be normalized out later.)

[0470] In such an instance, setting the D probability 1 in the top row has the effect of allowing the alignment to begin anywhere in the haplotype. It may also position an initial M transition into the second row, rather than permitting I transitions into the second row. Typically, I transitions may be permitted in the bottom row. In various instances, the initial 1.0 values may be put in M slots of the top row. Each other cell, however, may have its 3 probabilities computed from its 3 adjacent neighboring cells: above, left, and above-left. These 9 input probabilities may then contribute to the 3 result probabilities according to the state transition probabilities, and the sequence movement rules: transition to D horizontally, to I vertically, and to M diagonally.

[0471] This 3-to-1 computation dependency restricts the order that cells may be computed. They can be computed left to right in each row, progressing through rows from top to bottom, or top to bottom in each column, progressing rightward. Additionally, they may be computed in anti-diagonal wavefronts, where the next step is to compute all cells (n,m) where n+m equals the incremented step number. This wavefront order has the advantage that all cells in the anti-diagonal may be computed independently of each other. The bottom row of the matrix then, at the final read position, may be configured to represent the completed alignments. In such an instance, the Haplotype Caller will work by summing the I and M probabilities of all bottom row cells. In various embodiments, the system may be set up so that no D transitions are permitted within the bottom row, or a D transition probability of 0.0 may be used there, so as to avoid double counting.

[0472] As described herein, in various instances, each HMM evaluation may operate on a sequence pair, such as on a haplotype and a read pair. For instance, within a given active region, each of a set of haplotypes may be HMM-evaluated vs. each of a set of reads. In such an instance, the hardware input bandwidth may be reduced and/or minimized by transferring the set of reads and the set of haplotypes once, and letting HW generate the N×M pair operations. In certain instances, Smith-Waterman may be configured to queue up individual HMM operations, each with its own copy of read and haplotype data. This has the advantage of simplicity, low memory requirements, and flexibility if there is a need to perform other than precisely the N×M possible pairs.

[0473] There are three parallel multiplications (e.g., additions in log space), then two serial additions (~5-6 stage approximation pipelines), then an additional multiplication. In such an instance, the full pipeline may be about L=12-16 cycles long. The I & D calculations may be about half the length. The pipeline may be fed a multiplicity of input probabilities, such as 2 or 3 or 5 or 7 or more input probabilities each cycle, such as from one or more already computed neighboring cells (M and/or D from the left, M and/or I from above, and/or M and/or I and/or D from above-left). It may also include one or more haplotype bases, and/or one or more read bases such as with associated parameters, e.g., pre-processed parameters, each cycle. It outputs the M & I & D result set for one cell each cycle, after fall-through latency.

[0474] To keep the pipeline full, L independent cell calculations should be in progress at any one time. As can be seen with respect to FIG. **7**, these could of course be from separate HMM matrices **30**, but it is efficient for them to be along an anti-diagonal wavefront **35**.

[0475] As can be seen with respect to FIG. **8**, a difficulty is that the inputs to the pipeline for a new cell to compute come from one or more of its neighboring cells, such as its two or three neighboring cells of the matrix **30**, such as depicted in FIG. **7**.

[0476] In various instances, these neighboring cells in the matrix **30** can be computed as a variable, however such computations take a long time, which can become an issue with the time taken for storing and retrieving such intermediate results data. As can be seen with respect to FIG. **8**, a single cell in a matrix **30** pipeline can be configured such as by employing a horizontal swath of processing engines of one row high for each pipeline stage. In such an instance, the pipeline can follow an anti-diagonal within the swath, wrapping from the bottom to top of the swath, and wrapping the swath itself when the right edge of the matrix is reached, as depicted FIG. **9**.

[0477] The advantage of this configuration is that the 3 neighboring cells employed for a new calculation of an instant neighboring cell have recently been computed prior to computing the neighboring cell in the matrix **30**, such as a fixed number of cycles ago, as depicted in the FIG. **10**.

[0478] In various instances, current outputs at the pipeline's end are from a cell begun L cycles ago, so any time delays may be shortened by L, as depicted in FIG. **11**.

[0479] In various instances, there may be a delay, such as a one or more cycle delay, which delay may be just a register slice, such as where the L+1 delay may be a shift register or a shallow circular buffer. Results at the bottom of the swath may be stored in a local memory, and may be re-injected into the pipeline each time the position wraps vertically in the next swath. Dead cycles may or may not be required while the pipeline is wrapping horizontally from one swath to the next. For instance, if the input feed is controlled carefully, and left-column nulls are injected in the right clock cycles, a pipeline anti-diagonal in progress should be able to straddle between the right end of one swath and the left end of the next.

[0480] Further, in various instances, multiple cell computing pipelines can be configured to cooperate so as to achieve a high overall throughput. For example, there are ~65T cells that may be configured to compute for a whole genome, such as in a target of 15 minutes on the high-end. In such an instance, the pipelines can compute one cell per cycle at 300 MHz, and in such an instance 240 pipelines could be employed, which are a lot of pipelines. Theoretically, each of them could be working on a separate HMI matrix **30**, however, the amount of overhead logic to manage each matrix **30** will require additional resources, especially in the hardwired configuration, such as up to being multiplied by 240. In various instances, either of memory or logic could be a limiting factor. In such an instance, efficiency in the system may be enhanced such as by employing several pipelines that may be configured to cooperate with one another, so as to finish a single matrix **30** faster—if needed substantial management logic can be amortized.

[0481] To overcome any such limitations, the swath **35** cell order, as described above may be organized to make it easier for multiple pipelines to work on a single matrix. For instance, N pipelines could be configured to work on N swaths at a time, wherein each stays behind the compute wavefront **35** in the swath above. In such an instance, adjacent-swath **35**$_n$ pipelines may be configured so as to be synchronized, so that the lower one receives bottom-row results from the upper one at just the right moment, cutting down on memory requirements. To avoid N*L dead cycles at the start of each new matrix **35**$_n$, pipelines finishing their final swaths **35** in one matrix **30***a* can be configured to roll straight into upper swaths of the next matrix **30***b*.

[0482] Further, as illustrated in FIG. **12** is a histogram of HMM table dimensions, for 101-base reads. The left-to-right axis is haplotype length, the front-to-back axis is read length, and the vertical axis is log count.

[0483] From the high wall at the back, you can see the most common case by far is for the whole 101-base read to be used. This case represents about 35%, and the balance is distributed near evenly among lengths 10-100. The processed read length was not less than 10, in this instance. The high wall on the left is at haplotype length 41, about 5.4% of cases. Very few haplotypes were shorter, and the shortest was 9 bases. The longest haplotypes were 515 bases. The central plateau, from 136 bases to 349 bases, represents 87% of cases. The diagonal wall at the back-left is where haplotype length equals read length. Typically, the read sequence for HMM is clipped to the window length spanned by the haplotype, so it is rare for the read to be longer than the haplotype, and equal lengths are common. This distribution of matrix dimensions may contribute to a well-performing architecture, particularly if there are inefficiencies from dead cycles between matrices or swaths, uneven swath coverage, and the like.

[0484] In performing a variant call function, as disclosed herein, a De Bruijn Graph may be formulated, and when all of the reads in a pile up are identical, the DBG will be linear. However, where there are differences, the graph will form "bubbles" that are indicative of regions of differences resulting in multiple paths diverging from matching the reference alignment and then later re-joining in matching alignment. From this DBG, various paths may be extracted, which form candidate haplotypes, e.g., hypotheses for what the true DNA sequence may be on at least one strand, which hypotheses may be tested by performing an HMM, or modified HMM, operation on the data. Further still, a genotyping function may be employed such as where the possible diploid combinations of the candidate haplotypes may be formed, and for each of them, a conditional probability of observing the entire read pileup may be calculated. These results may then be fed into a Bayesian formula to calculate an absolute probability that each genotype is the truth, given the entire read pileup observed.

[0485] Hence, in accordance with the devices, systems, and methods of their use described herein, in various instances, a genotyping operation may be performed, which genotyping operation may be configured so as to be implemented in an optimized manner in software and/or in hardware. For instance, the possible diploid combinations of the candidate haplotypes may be formed, and for each combination, a conditional probability of observing the entire read pileup may be calculated, such as by using the constituent probabilities of observing each read given each haplotype from the pair HMM evaluation. The results of these calculations feed into a Bayesian formula so as to calculate an absolute probability that each genotype is the truth, given the entire read pileup observed.

[0486] Accordingly, in various aspects, the present disclosure is directed to a system for performing a haplotype or variant call operation on generated and/or supplied data so as to produce a variant call file with respect thereto. Specifically, as described herein above, in particular instances, a variant call file may be a digital or other such file that encodes the difference between one sequence and another, such as a the difference between a sample sequence and a reference sequence. Specifically, in various instances, the variant call file may be a text file that sets forth or otherwise details the genetic and/or structural variations in a person's genetic makeup as compared to one or more reference genomes.

[0487] For instance, a haplotype may be a set of genetic, e.g., DNA and/or RNA, variations, such as polymorphisms that reside in a person's chromosomes and as such may be passed on to offspring and thereby inherited together. Particularly, a haplotype can refer to a combination of alleles, e.g., one of a plurality of alternative forms of a gene such as may arise by mutation, which allelic variations are typically found at the same place on a chromosome. Hence, in determining the identity of a person's genome it is important to know which form of various different possible alleles a specific person's genetic sequence codes for. In particular instances, a haplotype may refer to one or more, e.g., a set, of nucleotide polymorphisms (e.g., SNPs) that may be found at the same position on the same chromosome.

[0488] Typically, in various embodiments, in order to determine the genotype, e.g., allelic haplotypes, for a subject, as described herein and above, a software based algorithm is engaged, such as an algorithm employing a haplotype call program, e.g., GATK, for simultaneously determining SNPs and/or insertions and/or deletions, i.e., indels, in an individual's genetic sequence. In particular, the algorithm may involve one or more haplotype assembly protocols such as for local de-novo assembly of a haplotype in one or more active regions of the genetic sequence being processed. Such processing typically involves the deployment of a processing function called a Hidden Markov Model (HMM) that is a stochastic and/or statistical model used to exemplify randomly changing systems such as where it is assumed that future states within the system depend only on the present state and not on the sequence of events that precedes it.

[0489] In such instances, the system being modeled bears the characteristics or is otherwise assumed to be a Markov process with unobserved (hidden) states. In particular instances, the model may involve a simple dynamic Bayesian network. Particularly, with respect to determining genetic variation, in its simplest form, there is one of four possibilities for the identity of any given base in a sequence being processed, such as when comparing a segment of a reference sequence, e.g., a hypothetical haplotype, and that of a subject's DNA or RNA, e.g., a read derived from a sequencer. However, in order to determine such variation, in a first instance, a subject's DNA/RNA must be sequenced, e.g., via a Next Gen Sequencer ("NGS"), to produce a readout or "reads" that identify the subject's genetic code. Next, once the subject's genome has been sequenced to produce one or more reads, the various reads, representative of the subject's DNA and/or RNA need to be mapped and/or aligned, as herein described above in great detail. The next step in the process then is to determine how the genes of the subject that have just been determined, e.g., having been mapped and/or aligned, vary from that of a prototypical reference sequence. In performing such analysis, therefore, it is assumed that the read potentially representing a given gene of a subject is a representation of the prototypical haplotype albeit with various SNPs and/or indels that are to presently be determined.

[0490] Accordingly, there exist commonly used software implementations for performing one or a series of such bioinformatics based analytical techniques so as to determine the various different genetic variations a subject may have in his or her genome. However, a common characteristic of such software based bioinformatics methods and systems employed for these purposes is that they are labor intensive, take a long time to execute on general purpose processors, and are prone to errors. A bioinformatics system, therefore, that could perform the algorithms or functions implemented by such software, e.g., various variant call functions, in a less labor and/or processing intensive manner with a greater percentage accuracy would be useful. However, the cost of analyzing, storing, and sharing this raw digital data has far outpaced the cost of producing it. This data analysis bottleneck is a key obstacle standing between these ever-growing raw data and the real medical insight we seek from it. The devices, systems, and methods of using the same, as presented herein, resolves these and other such needs in the art. Additionally, employing general purpose CPUs to perform specialized, repetitive mathematical computations are bulky, costly, and inefficient. So too, the power consumption, computation time, and physical footprint of an array of servers programmed to perform the HMM computations associated with the genome variant call operations, as disclosed herein, will all be undesirable compared to the traits of a system that performs such computations within a purpose-built, highly parallel microchip that is the subject of this disclosure.

[0491] Specifically, in particular aspects, devices, systems, and/or methods for practicing the same, such as for performing a haplotype and/or variant call function, such as deploying an HMM function, for instance, in an accelerated haplotype caller is provided. In various instances, in order to overcome these and other such various problems known in the art, the HMM accelerator herein presented may be configured to be operated in a manner so as to be implemented in software, implemented in hardware, or a combination of being implemented and/or otherwise controlled in part by software and/or in part by hardware. For instance, in a particular aspect, the disclosure is directed to a method by which data pertaining to the DNA and/or RNA sequence identity of a subject and/or how the subject's genetic information may differ from that of a reference genome may be determined.

[0492] In such an instance, the method may be performed by the implementation of a haplotype or variant call function, such as employing an HMM protocol. Particularly, the HMM function may be performed in hardware, such as on an accelerated device, in accordance with a method described herein. In such an instance, the hardware based HMI accelerator may be configured to receive and process the sequenced, mapped, and/or aligned data, to process the same, e.g., to produce a variant call file, as well as to transmit the processed data back throughout the system. Accordingly, the method may include deploying a system where data may be sent from a processor, such as a software-controlled CPU, to a haplotype caller implementing an accelerated HMM, which haplotype caller may be deployed on a microprocessor chip, such as an FPGA, ASIC, or structured ASIC. The method may further include the steps for processing the data to produce HMI result data, which results may then be fed back to the CPU.

[0493] Particularly, in one embodiment, as can be seen with respect to FIG. **13**, a variant call system **1** is provided. Specifically, FIG. **13** provides a high level view of an HMM interface structure. In particular embodiments, the variant call system **1** is configured to accelerate at least a portion of a variant call operation, such as an HMI operation. Hence, in various instances, a variant call system may be referenced herein as an HMI system **1**. The system **1** includes a server having one or more central processing units (CPU) **1000** configured for performing one or more routines related to the sequencing and/or processing of genetic information.

[0494] Additionally, the system **1** includes a peripheral device **2**, such as an expansion card, that includes a microchip **7**, such as an FPGA, ASIC, or sASIC. It is to be noted that the term ASIC may refer equally to a sASIC, where appropriate. The peripheral device **2** includes an interconnect **3** and a bus interface **4**, such as a parallel or serial bus, which connects the CPU **1000** with the chip **7**. For instance, the device **2** may comprise a peripheral component interconnect, such as a PCI, PCI-X, PCIe, or QPI, and may include a bus interface **4**, that is adapted to operably and/or communicably connect the CPU **1000** to the peripheral device **2**, such as for low latency, high data transfer rates. Accordingly, in particular instances, the interface may be a peripheral component interconnect express (PCIe) **4** that is associated with the microchip **7**, which microchip includes an HMM accelerator **8**. For example, in particular instances, the HMM accelerator **8** is configured for performing an accelerated HMM function, such as where the HMM function, in certain embodiments, may at least partially be implemented in the hardware of the FPGA, AISC, or sASIC.

[0495] Specifically, FIG. **13** presents a high-level figure of an HMM accelerator **8** having an exemplary organization of one or more engines **13**, such as a plurality of processing engines $\mathbf{13}a$-$\mathbf{13}_{m+1}$, for performing one or more processes of a variant call function, such as including an HMM task. Accordingly, the HMM accelerator **8** may be composed of a data distributor **9**, e.g., CentCom, and one or a multiplicity of processing clusters $\mathbf{11}$-$\mathbf{11}_{n+1}$ that may be organized as or otherwise include one or more instances **13**, such as where each instance may be configured as a processing engine, such as a small engine $\mathbf{13}a$-$\mathbf{13}_{m+1}$. For instance, the distributor **9** may be configured for receiving data, such as from the CPU **1000**, and distributing or otherwise transferring that data to one or more of the multiplicity of HMM processing clusters **11**.

[0496] Particularly, in certain embodiments, the distributor **9** may be positioned logically between the on-board PCIe interface **4** and the HMM accelerator module **8**, such as where the interface **4** communicates with the distributor **9** such as over an interconnect or other suitably configured bus **5**, e.g., PCIe bus. The distributor module **9** may be adapted for communicating with one or more HMM accelerator clusters **11** such as over one or more cluster buses **10**. For instance, the HMM accelerator module **8** may be configured as or otherwise include an array of clusters **11**a-**11**$_{n+1}$, such as where each HMM cluster **11** may be configured as or otherwise includes a cluster hub **11** and/or may include one or more instances **13**, which instance may be configured as a processing engine **13** that is adapted for performing one or more operations on data received thereby. Accordingly, in various embodiments, each cluster **11** may be formed as or otherwise include a cluster hub **11**a-**11**$_{n+1}$, where each of the hubs may be operably associated with multiple HMM accelerator engine instances **13**a-**13**$_{m+1}$, such as where each cluster hub **11** may be configured for directing data to a plurality of the processing engines **13**a-**13**$_{m+1}$ within the cluster **11**.

[0497] In various instances, the HMM accelerator **8** is configured for comparing each base of a subject's sequenced genetic code, such as in read format, with the various known haplotypes of a reference sequence and determining the probability that any given base at a position being considered either matches or doesn't match the relevant haplotype, i.e., the read includes an SNP, an insertion, or a deletion, thereby resulting in a variation of the base at the position being considered. Particularly, in various embodiments, the HMM accelerator **8** is configured to assign transition probabilities for the sequence of the bases of the read going between each of these states, Match ("M"), Insert ("I"), or Delete ("D") as described in greater detail herein below.

[0498] More particularly, dependent on the configuration, the HMM acceleration function may be implemented in either software, such as by the CPU **1000** and/or microchip **7**, and/or may be implemented in hardware and may be present within the microchip **7**, such as positioned on the peripheral expansion card or board **2**. In various embodiments, this functionality may be implemented partially as software, e.g., run by the CPU **1000**, and partially as hardware, implemented on the chip **7**. Accordingly, in various embodiments, the chip **7** may be present on the motherboard of the CPU **1000**, or it may be part of the peripheral device **2**, or both. Consequently, the HMM accelerator module **8** may include or otherwise be associated with various interfaces, e.g., 3, 5, 10, and/or 12 so as to allow the efficient transfer of data to and from the processing engines **13**.

[0499] Accordingly, as can be seen with respect to FIG. **13**, in various embodiments, a microchip **7** configured for performing a variant, e.g., haplotype, call function is provided. The microchip **7** may be associated with a CPU **1000** such as directly coupled therewith, e.g., included on the motherboard of a computer, or indirectly coupled thereto, such as being included as part of a peripheral device **2** that is operably coupled to the CPU **1000**, such as via one or more interconnects, e.g., 3, 4, 5, 10, and/or 12. In this instance, the microchip **7** is present on the peripheral device **2**.

[0500] Hence, the peripheral device **2** may include a parallel or serial expansion bus **4** such as for connecting the peripheral device **2** to the central processing unit (CPU) **1000** of a computer and/or server, such as via an interface **3**, e.g., DMA. In particular instances, the peripheral device **2** and/or serial expansion bus **4** may be a Peripheral Component Interconnect express (PCIe) that is configured to communicate with or otherwise include the microchip **7**, such as via connection **5**. As described herein, the microchip **7** may at least partially be configured as or may otherwise include an HMM accelerator **8**. The HMM accelerator **8** may be configured as part of the microchip **7**, e.g., as hardwired and/or as code to be run in association therewith, and is configured for performing a variant call function, such as for performing one or more operations of a Hidden Markov Model, on data supplied to the microchip **7** by the CPU **1000**, such as over the PCIe interface **4**. Likewise, once one or more variant call functions have been performed, e.g., one or more HMM operations run, the results thereof may be transferred from the HMM accelerator **8** of the chip **7** over the bus **4** to the CPU **1000**, such as via connection **3**.

[0501] For instance, in particular instances, a CPU **1000** for processing and/or transferring information and/or executing instructions is provided along with a microchip **7** that is at least partially configured as an HMM accelerator **8**. The CPU **1000** communicates with the microchip **7** over an interface **5** that is adapted to facilitate the communication between the CPU **1000** and the HMM accelerator **8** of the microchip **7** and therefore may communicably connect the CPU **1000** to the HMM accelerator **8** that is part of the microchip **7**. To facilitate these functions, the microchip **7** includes a distributor module **9**, which may be a CentCom, that is configured for transferring data to a multiplicity of HMM engines **13**, e.g., via one or more clusters **11**, where each engine **13** is configured for receiving and processing the data, such as by running an HMM protocol thereon, computing final values, outputting the results thereof, and repeating the same. In various instances, the performance of an HMM protocol may include determining one or more transition probabilities, as described herein below. Particularly, each HMM engine **13** may be configured for performing a job such as including one or more of the generating and/or evaluating of an HMM virtual matrix to produce and output a final sum value with respect thereto, which final sum expresses the probable likelihood that the called base matches or is different from a corresponding base in a hypothetical haplotype sequence, as described herein below.

[0502] FIG. **14** presents a detailed depiction of the HMM cluster **11** of FIG. **13**. In various embodiments, each HMM cluster **11** includes one or more HMM instances **13**. One or a number of clusters may be provided, such as desired in accordance with the amount of resources provided, such as on the chip. Particularly, a HMM cluster may be provided, where the cluster is configured as a cluster hub **11**. The cluster hub **11** takes the data pertaining to one or more jobs **20** from the distributor **9**, and is further communicably connected to one or more, e.g., a plurality of, HMM instances **13**, such as via one or more HMI instance busses **12**, to which the cluster hub **11** transmits the job data **20**.

[0503] The bandwidth for the transfer of data throughout the system may be relatively low bandwidth process, and once a job **20** is received, the system **1** may be configured for completing the job, such as without having to go off chip **7** for memory. In various embodiments, one job **20**a is sent to one processing engine **13**a at any given time, but several jobs **20**$_{a-n}$ may be distributed by the cluster hub **11** to several different processing engines **13**a-**13**$_{m+1}$, such as where each of the processing engines **13** will be working on a single job **20**, e.g., a single comparison between one or more reads and one or more haplotype sequences, in parallel and at high speeds. As described below, the performance of such a job **20** may typically involve the generation of a virtual matrix whereby the subject's "read" sequences may be compared to one or more, e.g., two, hypothetical haplotype sequences, so as to determine the differences there between. In such instances, a single job **20** may involve the processing of one or more matrices having a multiplicity of cells therein that need to be processed for each comparison being made, such as on a base by base basis. As the human genome is about 3 billion base pairs, there may be on the order of 1 to 2 billion different jobs to be performed when analyzing a 30× oversampling of a human genome (which is equitable to about 20 trillion cells in the matrices of all associated HMI jobs).

[0504] Accordingly, as described herein, each HMI instance **13** may be adapted so as to perform an HMI protocol, e.g., the generating and processing of an HMM matrix, on sequence data, such as data received thereby from the CPU **1000**. For example, as explained above, in sequencing a subject's genetic material, such as DNA, the DNA is broken down into segments, such as up to about 100 bases in length. The identity of these 100 base segments are then determined, such as by an automated sequencer, and "read" into a FASTQ text based file format that stores both each base identity of the read along with a Phred quality score (e.g., typically a number between 0 and 63 in log scale, where a score of 0 indicates the least amount of confidence that the called base is correct, with scores between 20 to 45 generally being acceptable as relatively accurate).

[0505] Particularly, as indicated above, a Phred quality score is a quality indicator that measures the quality of the identification of the nucleobase identities generated by the sequencing processor, e.g., by the automated DNA/RNA sequencer. Hence, each read base includes its own quality, e.g., Phred, score based on what the sequencer evaluated the quality of that specific identification to be. The Phred represents the confidence with which the sequencer estimates that it got the called base identity correct. This Phred score is then used by the implemented HMM module **8**, as described in detail below, to further determine the accuracy of each called base in the read as compared to the haplotype to which it has been mapped and/or aligned, such as by determining its Match, Insertion, and/or Deletion transition probabilities, e.g., in and out of the Match state. It is to be noted that in various embodiments, the system **1** may modify or otherwise adjust the initial Phred score prior to the performance of an HMM protocol thereon, such as by taking into account neighboring bases/scores and/or fragments of neighboring DNA and allowing such factors to influence the Phred score of the base, e.g., cell, under examination.

[0506] In such instances, as can be seen with respect to FIG. **15**, the system **1**, e.g., computer software, may determine and identify various active regions **500**$_n$ within the sequenced genome that may be explored and/or otherwise subjected to further processing as herein described, which may be broken down into jobs **20***n* that may be parallelized amongst the various cores and available threads **1007** throughout the system **1**. For instance, such active regions **500** may be identified as being sources of variation between the sequenced and reference genomes. Particularly, the CPU **1000** may have multiple threads **1007** running, identifying active regions **500***a*, **500***b*, and **500***c*, compiling and aggregating various different jobs **20**$_n$ to be worked on, e.g., via a suitably configured aggregator **1008**, based on the active region(s) **500***a-c* currently being examined. Any suitable number of threads **1007** may be employed so as to allow the system **1** to run at maximum efficiency, e.g., the more threads present the less active time spent waiting.

[0507] Once identified, compiled, and/or aggregated, the threads **1007/1008** will then transfer the active jobs **20** to the data distributor **9**, e.g., CentCom, of the HMM module **8**, such as via PCIe interface **4**, e.g., in a fire and forget manner, and will then move on to a different process while waiting for the HMM **8** to send the output data back so as to be matched back up to the corresponding active region **500** to which it maps and/or aligns. The data distributor **9** will then distribute the jobs **20** to the various different HMM clusters **11**, such as on a job-by-job manner. If everything is running efficiently, this may be on a first in first out format, but such does not need to be the case. For instance, in various embodiments, raw jobs data and processed job results data may be sent through and across the system as they become available.

[0508] Particularly, as can be seen with respect to FIG. **15**, the various job data **20** may be aggregated into 4K byte pages of data, which may be sent via the PCIe **4** to and through the CentCom **9** and on to the processing engines **13**, e.g., via the clusters **11**. The amount of data being sent may be more or less than 4K bytes, but will typically include about 100 HMM jobs per 4K (e.g., 1024) page of data. Particularly, these data then get digested by the data distributor **9** and are fed to each cluster **11**, such as where one 4K page is sent to one cluster **11**. However, such need not be the case as any given job **20** may be sent to any given cluster **11**, based on the clusters that become available and when. Accordingly, as can be seen with respect to FIGS. **17** and **18**, each job **20** may have a job ID that accompany each job, which job ID flows through the overall process substantially unmodified so the system, e.g., software and/or hardware, can use those identifications so that it can be maintained to which active region **500** each particular job **20** and/or result refers.

[0509] Accordingly, the cluster **11** approach as presented here efficiently distributes incoming data to the processing engines **13** at high-speed. Specifically, as data arrives at the PCIe interface **4** from the CPU **1000**, e.g., over DMA connection **3**, the received data may then be sent over the PCIe bus **5** to the CentCom distributor **9** of the variant caller microchip **7**. The distributor **9** then sends the data to one or more HMM processing clusters **11**, such as over one or more cluster dedicated buses **10**, which cluster **11** may then transmit the data to one or more processing instances **13**, e.g., via one or more instance buses **12**, such as for processing. In this instance, the PCIe interface **4** is adapted to provide data through the peripheral expansion bus **5**, distributor **9**, and/or cluster **10** and/or instance **12** busses at a rapid rate, such as at a rate that can keep one or more, e.g., all, of the HMM accelerator instances **13**$_{a-(m+1)}$ within one or more, e.g., all, of the HMM clusters **11**$_{a-(n+1)}$ busy, such as over a prolonged period of time, e.g., full time, during the period over which the system **1** is being run, the jobs **20** are being processed, and whilst also keeping up with the output of the processed HMM data that is to be sent back to one or more CPUs **1000**, over the PCIe interface **4**.

[0510] For instance, any inefficiency in the interfaces 3, 5, 10, and/or 12 that leads to idle time for one or more of the HMM accelerator instances **13** may directly add to the overall processing time of the system **1**. Particularly, when analyzing a human genome, there may be on the order of two or more billion different jobs **20** that need to be distributed to the various HMM clusters **11** and processed over the course of a time period, such as under 1 hour, under 45 minutes, under 30 minutes, under 20 minutes including 15 minutes, 10 minutes, 5 minutes, or less.

[0511] For example, each typical job **20** may have on the order of a few hundred bytes of write data associated with it. In such an instance, the total amount of write data may be on the order of several hundred Gigabytes to one or more thousand of Gigabytes, such as over 1 Terabyte of data, such as over the course of processing a whole genome. However, in an instance such as this, the data to be fed back to the CPU **1000** may be as little as 16-bytes per job **20**. Hence, there is a need for efficient data distribution and collection, which need may not arise as much from the amount of data (~1.1 Gbyte/s average write rate, ~64 Mbyte/s average read rate), as from the requirement that the data be sliced up and parsed out to (or collected from) one or more of the various parallel jobs **20** being performed by the one or more clusters **11** and/or one or more instances **13**.

[0512] More particularly, if it is assumed that 200 MHz is the speed of the clock associated with the Cluster Buses **10** and a data width of 32 bits is moving through the bus of each HMM cluster **11** during each clock cycle, as described in detail below, then something on the order of six HMM clusters **11***a-f* will provide a data write data bandwidth capability that exceeds the ~1.1 GB/sec average requirement, such as by a factor of four, or greater. Accordingly, in one exemplary embodiment, an initial configuration for the Cluster Buses **10** may involve a 200 MHz clock and data transfer rate as well as six HMM clusters **11***a-f*. However, as routing and/or throughput requirements evolve, the number of clusters **11** or the speed for the Cluster Buses **10** may be adjusted, so the cluster count and Cluster Bus **10** speed be may be parametrize-able so as to meet evolving needs.

[0513] Accordingly, FIGS. **14** and **15** sets forth an overview of the data flow throughout the software and/or hardware of the system **1**, as described generally above. As can be seen with respect to FIGS. **14** and **15**, the system **1** may be configured in part to transfer data, such as between the PCIe interface **4** and the distributor **9**, e.g., CentCom, such as over the PCIe bus **5**. Additionally, the system **1** may further be configured in part to transfer the received data, such as between the distributor **9** and the one or more HMM clusters **11**, such as over the one or more cluster buses **10**. Hence, in various embodiments, the HMM accelerator **8** may include one or more clusters **11**, such as one or more clusters **11** configured for performing one or more processes of an HMM function. In such an instance, there is an interface, such as a cluster bus **10**, that connects the CentCom **9** to the HMM cluster **11**.

[0514] For instance, FIGS. **14** and **16** is a high level diagram depicting the interface in to and out of the HMM module **8**, such as into and out of a cluster module. As can be seen with respect to FIG. **16**, each HMM cluster **11** may be configured to communicate with, e.g., receive data from and/or send final result data, e.g., sum data, to the CentCom data distributor **9** through a dedicated cluster bus **10**. Particularly, any suitable interface or bus **5** may be provided so long as it allows the PCIe interface **4** to communicate with the data distributor **9**. More particularly, the bus **5** may be an interconnect that includes the interpretation logic useful in talking to the data distributor **9**, which interpretation logic may be configured to accommodate any protocol employed to provide this functionality. Specifically, in various instances, the interconnect may be configured as a PCIe bus **5**. Additionally, the cluster **11** may be configured such that single or multiple clock domains may be employed therein, and hence, one or more clocks may be present within the cluster **11**. In particular instances, multiple clock domains will be provided. For example, a slower clock may be provided, such as for communications, e.g., to and from the cluster **11**. Additionally, a faster, e.g., a high speed, clock may be provided which may be employed by the HMM instances **13** for use in performing the various state calculations described herein.

[0515] Particularly, in various embodiments, as can be seen with respect to FIG. **16**, the system **1** may be set up such that, in a first instance, as the data distributor **9** leverages the existing CentCom IP, a collar, such as a gasket, may be provided, where the gasket is configured for translating signals to and from the CentCom interface **5** from and to the HMM cluster interface or bus **10**. For instance, an HMM cluster bus **10** may communicably and/or operably connect the CPU **1000** to the various clusters **11** of the HMM accelerator module **8**.

[0516] Hence, as can be seen with respect to FIG. **16**, structured write and/or read data for each haplotype and/or for each read may be sent throughout the system **1**. Particularly, as can be seen with respect to FIG. **17**, an exemplary write data structure **22** is provided, such as where the data structure may include one or more, e.g., a plurality, of 32 bit words, such as on a top layer that function as control words and/or contain the haplotype length and/or other control data, e.g., in the reserved area. The next layer of data may also be a 32 bit word such as includes the haplotype ID, which ID may be used by the system software to take the output results and correlate them back to where it came from in the associated active region being processed. With respect to analyzing the haplotype sequence, 8-four bit bases may be provided for each 32 bit word, and two haplotype sequences may be analyzed at a given time, e.g., thereby filling layers 3 and 4 of the data structure. It is to be noted that the word layers need not be 32 bits, but in various instances, the use of a 32-bit word may be particularly efficient.

[0517] Accordingly, with respect to the transfer of write data, one or more, e.g., each, HMM engine instance **13** within or otherwise associated with the HMM cluster hub **11** may be configured to include or otherwise be operably connected with one, two, or more separate one or two-port memories, such as 1 read port and/or 1 write port memory. These memories may be a HMEM **16** and/or an RMEM **18**, such as where each memory includes both a read and a write port. FIG. **17** exemplifies the possible contents of a single HMEM data structure **22**, while FIG. **18**, as explained below, exemplifies the possible contents of a single RMEM data structure **24**. In such instances, the data distributor **9** may be configured to access the write port, and the HMM engine instance **13** may be configured to access the read port of the HMEM and RMEM memories.

[0518] Specifically, in various instances, one or more of the interfaces, such as the cluster bus interface **10** may be associated with a clock, such as a cluster bus interface clock, which may be run at a relatively slower cycle speed. Additionally, various other components of the system **1**, e.g., the HMM instance **13**, may be associated with one or more other clocks of the system, such as a core domain clock, which clock may be run at a relatively faster cycle speed. In such instances, therefore, the write port on both the HMEM **16** and the RMEM **18** may be connected to the cluster bus interface clock, while the read port on both the HMEM **16**

and the RMEM **18** may be connected to the HMM engine core clock domain. Consequently, these memories may form a synchronous or an asynchronous boundary between the slower cluster bus interface clock domain and the faster HMM engine core clock domain.

[0519] Additionally, as shown with respect to FIG. **17**, the HMEM **16** may be used to hold the reference haplotype base identifier and other related control information. Each reference haplotype base identifier may be represented within the data structure **22** as four bits, such as by using a mapping scheme such as: 0 implies haplotype base is "A;" 1 implies haplotype base is "C;" 2 implies haplotype base is "G;" 3 implies haplotype base is "T;" and 15 implies haplotype base is "N." It is to be noted that other various sequences and combinations of coding for the same may be employed without departing form the nature of this embodiment. Accordingly, in particular instances, A, C, G, and T, may be defined as 0, 1, 2, and 3, and where there is an "N" base, e.g., where the reference cannot make a good call as to the identity of a particular base, it may be defined as **15**. All other four-bit values may be RESERVED. It is to be noted that each HMM engine instance **13** may have one, two, or more logical HMEM instances. Also note that bits [31:30] of the first word of each haplotype record may be written as "10" binary.

[0520] As indicated, these haplotype base identifiers may be packed as eight 4-bit values per 32-bit write word, with base identifiers corresponding to earlier values in the reference sequence being located closer to bit 0 of the 32 bit word (see FIG. **17**, for more information on the packing scheme). Accordingly, enough space is provisioned in the HMEM to hold one, two, or more complete reference sequences per HMM job **20**, and these complete sequences may be thought of as being held in separate logical HMEM instances. This allows better use of both interface **4** and HMM engine **13** resources, as a read sequence that is to be compared to one or more, e.g., multiple, different reference haplotype sequences need only be written to an HMM engine instance **13** once.

[0521] In addition to the reference haplotype base identifiers, the HMEM may also contain a haplotype length field, and a 32-bit haplotype ID. For example, the haplotype length field communicates the length of the reference haplotype sequence. The haplotype ID may be a value generated by the variant call software of the CPU **1000**, e.g., a thread **1007** thereof, and may be included with the final output sum that is fed back to the CPU **1000**. Such "Hap ID" may therefore be used by the variant call software of the system **1** to associate a final HMM sum output with a specific reference haplotype. For instance, in various instances, different jobs **20** may take different amounts of time to complete, so there is no guarantee that the order in which the thread **1007** issues the jobs **20** to the hardware accelerator **8** will be the order in which it will receive the results back from those jobs.

[0522] As can be seen with respect to FIG. **18**, an exemplary read data structure **24** is provided, such as where the data structure may include one or more 32 bit words, such as on the top layer that function as control words and/or contain the read length, job-specific control information and/or other control data, e.g., in the reserved area. These data may include instructions regarding specific parameters directing the software to perform certain calculations so that the hardware need not calculate them. Such data could be calculated by the hardware but it may in certain instances be more efficient to perform such tasks in software as they need only be calculated once per job.

[0523] The next layer of data may also be a 32 bit word such as includes the read ID, which when taken with the haplotype ID defines what the job **20** is and where it is from in the associated active region **500** being processed. With respect to analyzing the read sequence, for each read base the Phred quality score may be provided and a gap open penalty (GOP), as explained below, may be provided, both of which may be in 6-bits. It is to be noted that the read memory **18** may be deeper than the haplotype memory for a given sequence length, and this is in part because instead of simply including 8 bases per 32-bit word, only 2 bases per 32-bit road may be used, since the Phred score and GOP is also included. Again, it is to be noted that the word layers need not be 32 bits, but in various instances, the use of a 32-bit word may be particularly efficient. In various embodiments, the HMEM **16** and RMEM **18** may be configured so as to have enough space to hold the data associated with a haplotype or read sequence(s) up to a length of 1000 or more, such as 1020 or more, such as 1050 or 1080 or more bases. Of course, shorter or longer sequences could be tolerated with the corresponding increase in memory-dedicated resources.

[0524] Accordingly, the data structure associated with each read base is set forth in FIG. **18**. In this instance, a 2-bit base identifier, with a {0,1,2,3} specifies {A,C,G,T}, respectively. Further, a 6-bit base quality may be present in Phred space (where a quality=0 or other determined base quality is used to imply a base identifier of "N") as well as a 6-bit insertion/deletion gap open penalty. Accordingly, the data associated with the two read bases may be packed into each 32-bit word that is delivered to the HMM cluster **11**, with read base information corresponding to earlier values in the read sequence being located in the lower half of the 32-bit word (see FIG. **6** for more information on the packing scheme).

[0525] In addition to the read base identifiers, per-read-base quality information, and per-read-base gap open penalty, the RMEM **18** may also contain the read length field, the job-specific control information field, and a 32-bit read ID. The read length field can be configured to communicate the length of the read sequence. The read ID may be a value generated by the CPU **1000**, or a thread **1007** thereof, which may be included with the final output sum to be fed back to the CPU **1000**. This "Read ID" may be used by the system **1** to associate a final HMM sum output with a specific reference read sequence (as before, it is noted that different jobs may take different amounts of time, so there is no guarantee that the order in which the CPU **1000** issues the jobs is the order in which it will receive the results from those jobs).

[0526] Accordingly, when each HMI engine instance **13** completes a job, a 128-bit record is made available to the data distributor **9** for reading. In order to efficiently utilize the interface **4**, e.g., PCIe interface, and associated bandwidth, the data distributor **9** may collect records from multiple completed jobs **20**$_n$ before sending the data upstream to the CPU **1000**. The record associated with each completed job **20** may contain the following information: Job Status Word, Hap ID, Read ID, and the Final HMI Sum Value. Accordingly, when the computing has been completed, there are 4-32 bit words that are then returned to the variant call software of the CPU **1000**, the status word characterizes the job **20**, the haplotype and read IDs map the job **20** back to its corresponding active region **500**, and the final sum value, is described in greater detail below.

[0527] For instance, the Read ID and Hap ID are typically those 32 bit values that the CPU **1000**, or thread **1007** thereof, provides in the write stream to use in identifying job **20** results. Since, the jobs may not complete in the order that they were issued, the Read and Hap IDs are the mechanism the system **1** uses to properly associate jobs with results. The final HMM sum value may be a 32-bit value that is the output of the HMI matrix computation and summing process, described below. This value may be in a variant of floating point format, such as with a number of mantissa and exponent bits that are programmable.

[0528] Following a job **20** being input into the HMI engine, an HMM engine **13** may typically start either: a) immediately, if it is IDLE, or b) after it has completed its currently assigned task. It is to be noted that each HMI accelerator engine **13** can handle ping and pong inputs (e.g., can be working on one data set while the other is being loaded), thus minimizing downtime between jobs. Additionally, the HMM cluster collar **11** may be configured to automatically take the input job **20** sent by the data distributor **9** and assign it to one of the HMM engine instances **13** in the cluster **11** that can receive a new job. There need not be a control on the software side that can select a specific HMM engine instance **13** for a specific job **20**. However, in various instances, the software can be configured to control such instances.

[0529] Accordingly, in view of the above, the system **1** may be streamlined when transferring the results data back to the CPU, and because of this efficiency there is not much data that needs to go back to the CPU to achieve the usefulness of the results. This allows the system to achieve about a 30 minute or less, such as about a 25 or about a 20 minute or less, for instance, about a 18 or about a 15 minute or less, including about a 10 or about a 7 minute or less, even about a 5 or about a 3 minute or less variant call operation, dependent on the system configuration.

[0530] FIG. **19** presents a high-level view of various functional blocks within an exemplary HMM engine **13** within a hardware accelerator **8**, on the FPGA or ASIC **7**. Specifically, within the hardware HMM accelerator **8** there are multiple clusters **11**, and within each cluster **11** there are multiple engines **13**. FIG. **19** presents a single instance of an HMM engine **13**. As can be seen with respect to FIG. **19**, the engine **13** may include an instance bus interface **12**, a plurality of memories, e.g., an HMEM **16** and an RMEM **18**, various other components **17**, HMM control logic **15**, as well as a result output interface **19**. Particularly, on the engine side, the HMM instance bus **12** is operably connected to the memories, HMEM **16** and RMEM **18**, and may include interface logic that communicates with the cluster hub **11**, which hub is in communications with the distributor **9**, which in turn is communicating with the PCIe interface **4** that communicates with the variant call software being run by the CPU and/or server **1000**. The HMM instance bus **12**, therefore, receives the data from the CPU **1000** and loads it into one or more of the memories, e.g., the HMEM and RMEM.

[0531] In such an instance, enough memory space should be allocated such that at least one or two or more haplotypes, e.g., two haplotypes, may be loaded, e.g., in the HMEM **16**, per given read sequence that is loaded, e.g., into the RMEM **18**, which when multiple haplotypes are loaded results in an easing of the burden on the PCIe bus **5** bandwidth. In particular instances, two haplotypes and two read sequences may be loaded into their respective memories, which would allow the four sequences to be processed together in all relevant combinations. In other instances four, or eight, or sixteen sequences, e.g., pairs of sequences, may be loaded, and in like manner be processed in combination, such as to further ease the bandwidth when desired.

[0532] Additionally, enough memory may be reserved such that a ping-pong structure may be implemented therein such that once the memories are loaded with a new job **20***a*, such as on the ping side of the memory, a new job signal is indicated, and the control logic **15** may begin processing the new job **20***a*, such as by generating the matrix and performing the requisite calculations, as described herein and below. Accordingly, this leaves the pong side of the memory available so as to be loaded up with another job **20***b*, which may be loaded therein while the first job **20***a* is being processed, such that as the first job **20***a* is finished, the second job **20***b* may immediately begin to be processed by the control logic **15**.

[0533] In such an instance, the matrix for job **20***b* may be preprocessed so that there is virtually no down time, e.g., one or two clock cycles, from the ending of processing of the first job **20***a*, and the beginning of processing of the second job **20***b*. Hence, when utilizing both the ping and pong side of the memory structures, the HMEM **16** may typically store 4 haplotype sequences, e.g., two a piece, and the RMEM **18** may typically store 2 read sequences. This ping-pong configuration is useful because it simply requires a little extra memory space, but allows for a doubling of the throughput of the engine **13**.

[0534] During and/or after processing the memories **16**, **18** feed into the transition probabilities calculator and lookup table (LUT) block **17***a*, which is configured for calculating various information related to "Priors" data, as explained below, which in turn feeds the Prior results data into the M, I, and D state calculator block **17***b*, for use when calculating transition probabilities. One or more scratch RAMs **17***c* may also be included, such as for holding the M, I, and D states at the boundary of the swath, e.g., the values of the bottom row of the processing swath, which as indicated, in various instances, may be any suitable amount of cells, e.g., about 10 cells, in length so as to be commensurate with the length of the swath **35**.

[0535] Additionally included is a separate results output interface block **19** so when the sums are finished they, e.g., the 4 32-bit words, can immediately be transmitted back to the variant call software of the CPU **1000**. It is to be noted that this configuration may be adapted so that the system **1**, specifically the M, I, and D calculator **17***b* is not held up waiting for the output interface **19** to clear, e.g., so long as it does not take as long to clear the results as it does to perform the job **20**. Hence, in this configuration, there may be three pipeline steps functioning in concert to make an overall systems pipeline, such as loading the memory, performing the MID calculations, and outputting the results. Further, it is noted that any given HMM engine **13** is one of many with their own output interface **19**, however they may share a common interface **10** back to the data distributor **9**. Hence, the cluster hub **11** will include management capabilities to manage the transfer ("xfer") of information through the HMM accelerator **8** so as to avoid collisions.

[0536] Accordingly, the following discussion goes into detail as to the processes being performed within each module of the HMM engines **13** as it receives the haplotype and read sequence data, processes it, and outputs results data pertaining to the same, as generally outlined above. Specifically, the high-bandwidth computations in the HMM engine **13**, within the HMM cluster **11**, are directed to computing and/or updating the match (M), insert (I), and delete (D) state values, which are employed in determining whether the particular read being examined matches the haplotype reference as well as the extent of the same, as described above. Particularly, the read along with the Phred score anf GOP value for each base in the read is transmitted to the cluster **11** from the distributor **9** and is thereby assigned to a particular processing engine **13** for processing. These data are then used by the M, I, and D calculator **17** of the processing engine **13** to determine whether the called base in the read is more or less likely to be correct and/or to be a match to its respective base in the haplotype, or to be the product of a variation, e.g., an insert or deletion; and/or if there is a variation, whether such variation is the likely result of a true variability in the haplotype or rather an artifact of an error in the sequence generating and/or mapping and/or aligning systems.

[0537] As indicated above, a part of such analysis includes the MID calculator **17** determining the transition probabilities from one base to another in the read going from one M, I, or D state to another in comparison to the reference, such as from a matching state to another matching state, or a matching state to either an insertion state or to a deletion state. In making such determinations each of the associated transition probabilities is determined and considered when evaluating whether any observed variation between the read and the reference is a true variation and not just some machine or processing error. For these purposes, the Phred score for each base being considered is useful in determining the transition probabilities in and out of the match state, such as going from a match state to an insert or deletion, e.g., a gapped, state in the comparison. Likewise, the transition probabilities of continuing a gapped state or going from a gapped state, e.g., an insert or deletion state, back to a match state are also determined. In particular instances, the probabilities in or out of the delete or insert state, e.g., exiting a gap continuation state, may be a fixed value, and may be referenced herein as the gap continuation probability or penalty. Nevertheless, in various instances, such gap continuation penalties may be floating and therefore subject to change dependent on the accuracy demands of the system configuration.

[0538] Accordingly, as depicted with respect to FIGS. **20** and **21** each of the M, I, and D state values are computed for each possible read and haplotype base pairing. In such an instance, a virtual matrix **30** of cells containing the read sequence being evaluated on one axis of the matrix and the associated haplotype sequence on the other axis may be formed, such as where each cell in the matrix represents a base position in the read and haplotype reference. Hence, if the read and haplotype sequences are each 100 bases in length, the matrix **30** will include 100 by 100 cells, a given portion of which may need to be processed in order to determine the likelihood and/or extent to which this particular read matches up with this particular reference. Hence, once virtually formed, the matrix **30** may then be used to determine the various state transitions that take place when moving from one base in the read sequence to another and comparing the same to that of the haplotype sequence, such as depicted in FIGS. **20** and **21**. Specifically, the processing engine **13** is configured such that a multiplicity of cells may be processed in parallel and/or sequential fashion when traversing the matrix with the control logic **15**. For instance, as depicted in FIG. **20**, a virtual processing swath **35** is propagated and moves across and down the matrix **30**, such as from left to right, processing the individual cells of the matrix **30** down the right to left diagonal.

[0539] More specifically, as can be seen with respect to FIG. **20**, each individual virtual cell within the matrix **30** includes an M, I, and D state value that needs to be calculated so as to asses the nature of the identity of the called base, and as depicted in FIG. **20** the data dependencies for each cell in this process may clearly be seen. Hence, for determining a given M state of a present cell being processed, the Match, Insert, and Delete states of the cell diagonally above the present cell need to be pushed into the present cell and used in the calculation of the M state of the cell presently being calculated (e.g., thus, the diagonal downwards, forwards progression through the matrix is indicative of matching).

[0540] However, for determining the I state, only the Match and Insert states for the cell directly above the present cell need be pushed into the present cell being processed (thus, the vertical downwards "gapped" progression when continuing in an insertion state). Likewise, for determining the D state, only the Match and Delete states for the cell directly left of the present cell need be pushed into the present cell (thus, the horizontal cross-wards "gapped" progression when continuing in a deletion state). As can be seen with respect to FIG. **20**, after computation of cell 1 (the shaded cell in the top most row) begins, the processing of cell 2 (the shaded cell in the second row) can also begin, without waiting for any results from cell 1, because there is no data dependencies between this cell in row 2 and the cell of row 1 where processing begins. This forms a reverse diagonal **35** where processing proceeds downwards and to the left, as shown by the red arrow. This reverse diagonal **35** processing approach increases the processing efficiency and throughput of the overall system. Likewise, the data generated in cell 1, can immediately be pushed forward to the cell down and forward to the right of the top most cell 1, thereby advancing the swath **35** forward.

[0541] For instance, FIG. **20** depicts an exemplary HMM matrix structure **35** showing the hardware processing flow. The matrix **35** includes the haplotype base index, e.g., containing 36 bases, positioned to run along the top edge of the horizontal axis, and further includes the base read index, e.g., 10 bases, positioned to fall along the side edge of the vertical axis in such a manner to from a structure of cells where a selection of the cells may be populated with an M, I, and D probability state, and the transition probabilities of transitioning from the present state to a neighboring state. In such an instance, as described in greater detail above, a move from a match state to a match state results in a forwards diagonal progression through the matrix **30**, while moving from a match state to an insertion state results in a vertical downwards progressing gap, and a move from a match state to a deletion state results in a horizontal progressing gap. Hence, as depicted in FIG. **21**, for a given cell, when determining the match, insert, and delete states for each cell, the match, insert, and delete probabilities of its three adjoining cells are employed.

[0542] The downwards arrow in FIG. **20** represents the parallel and sequential nature of the processing engine(s) that are configured so as to produce a processing swath or wave **35** that moves progressively along the virtual matrix in accordance with the data dependencies, see FIGS. **20** and **21**, for determining the M, I, and D states for each particular cell in the structure **30**. Accordingly, in certain instances, it may be desirable to calculate the identities of each cell in a downwards and diagonal manner, as explained above, rather than simply calculating each cell along a vertical or horizontal axis exclusively, although this can be done if desired. This is due to the increased wait time, e.g., latency, that would be required when processing the virtual cells of the matrix **35** individually and sequentially along the vertical or horizontal axis alone, such as via the hardware configuration.

[0543] For instance, in such an instance, when moving linearly and sequentially through the virtual matrix **30**, such as in a row by row or column by column manner, in order to process each new cell the state computations of each preceding cell would have to be completed, thereby increasing latency time overall. However, when propagating the M, I, D probabilities of each new cell in a downwards and diagonal fashion, the system **1** does not have to wait for the processing of its preceding cell, e.g., of row one, to complete before beginning the processing of an adjoining cell in row two of the matrix. This allows for parallel and sequential processing of cells in a diagonal arrangement to occur, and further allows the various computational delays of the pipeline associated with the M, I, and D state calculations to be hidden. Accordingly, as the swath **35** moves across the matrix **30** from left to right, the computational processing moves diagonally downwards, e.g., towards the left (as shown by the arrow in FIGS. **7** and **20**). This configuration may be particularly useful for hardware implementations, such as where the memory and/or clock-by-clock latency are a primary concern.

[0544] However, when implementing an HMM function, as herein described, in software, the memory and/or clock-by-clock latency concerns are secondary. Hence, when running an HMM function, as herein described, in software, a nested "for" loop process may be implemented. For instance, when implemented in software, the code may be configured so as to calculate all the possible state values in the virtual HMM matrix such as exemplified herein: "for haplotype_index=0 to (haplotype_length−1); for read_index=0 to (read_length−1); Update M, I, and D state values for (haplotype_index,read_index) base pairing; end. end." In its essence, this code instructs the system to go from beginning to end, such as going from the beginning of the row to the end, and/or from the beginning of the column to the end, looping down the rows and/or across the columns, or vice versa, all the way from the beginning to the end. Accordingly, where latency timing is not an issue, the system can simply begin at the first available bases in each of the haplotype and read sequence indices, compare them with one another to determine a match or mismatch probability, and then move to a comparison of the next subsequent base in the sequences to update the probabilities accordingly. In such an instance, a downwards diagonal processing swath need not be promulgated.

[0545] However, this row-by-row, column-by-column computation of the HMM states, as determined by the referenced exemplary code above, may not be as useful when providing an accelerator that is at least partially implemented in hardware. Particularly, where clock cycles are important and latencies thereof must be managed to achieve maximal efficiency, the swath based processing configuration of FIGS. **20** and **21** may be particularly useful. For example, there may be a one or more, such as a ten or twenty or more, such as a twenty five or fifty or more cycle latency to calculate any given state, and so the system can be configured so as to push more data into the cells of the matrix during such latency periods instead of just waiting around and doing nothing during such latency periods, thereby increasing throughput without affecting accuracy.

[0546] Hence, as can be seen with respect to FIGS. **20** and **21**, new data may be pushed into the system every single clock cycle, even though the pipeline itself may take ten or twenty or more clock cycles to complete its processing of any particular state of a given cell or group of cells. Particularly, if the pipeline delay through the M, I, and D state calculation, e.g., the clock cycle latencies thereof, is known, the processing of the matrix **30** may be configured, e.g., the processing swath **35** length adapted, such that by the time that the first, e.g., top, cell of the swath **35***a* is done being calculated, the system loops around and the beginning of the processing of the next swath **35***b* may be initiated, as described in greater detail with respect to FIG. **29**.

[0547] Accordingly, the length of the swath **35** may be configured so as to correlate with the latency of the clock cycles needed to determine the state values for given selection of cells. An increased latency period therefore would result in an increased number of cells being processed within any given length of swath **35**, and vice-versa with respect to decreased latency times. This then reduces the need and/or storing times for results data, such as in FIFO memories. Again, such a configuration is particularly useful in hardware implementations where memory resources and lookup times are important considerations. A further advantage of such hardware implementations is that the processing of such matrices 30_n may be performed in a highly parallelized manner, e.g., such as tens to hundreds to thousands of matrices being processed all at the same time performing various different read to haplotype comparisons, which cannot easily be achieved by employing core computing facilities running various known software implementations.

[0548] In these configurations, the actual value output from each call of an HMM engine **13**, e.g., after having calculated the entire matrix **30**, may be a bottom row (e.g., Row **35** of FIG. **21**) containing M, I, and D states, where the M and I states may be summed (the D states may be ignored at this point having already fulfilled their function in processing the calculations above), so as to produce a final sum value that may be a single probability that estimates, for each read and haplotype index, the probability of observing the read, e.g., assuming the haplotype was the true original DNA sampled.

[0549] Particularly, the outcome of the processing of the matrix **30**, e.g., of FIGS. **6** and **29**, may be a single value representing the probability that the read is an actual representation of that haplotype. This probability is a value between 0 and 1 and is formed by summing all of the M and I states from the bottom row of cells in the HMM matrix **30**. Essentially, what is being assessed is the possibility that something could have gone wrong in the sequencer, or associated DNA preparation methods prior to sequencing, so as to incorrectly produce a mismatch, insertion, or deletion into the read that is not actually present within the subject's genetic sequence. In such an instance, the read is not a true reflection of the subject's actual DNA.

[0550] Hence, accounting for such production errors, it can be determined what any given read actually represents with respect to the haplotype, and thereby allows the system to better determine how the subject's genetic sequence, e.g., en masse, may differ from that of a reference sequence. For instance, many haplotypes may be run against many read sequences, generating scores for all of them, and determining based on which matches have the best scores, what the actual genomic sequence identity of the individual is and/or how it truly varies from a reference genome.

[0551] More particularly, FIG. **21** depicts an enlarged view of a portion of the HMM state matrix **30** from FIG. **20**. As shown in FIG. **21**, given the internal composition of each cell in the matrix **30**, as well as the structure of the matrix as a whole, the M, I, and D state probability for any given "new" cell being calculated is dependent on the M, I, and D states of several of its surrounding neighbors that have already been calculated. Particularly, as shown in greater detail with respect to FIGS. **6** and **21**, in an exemplary configuration, there may be an approximately a 0.9998 probability of going from a match state to another match state, and there may be only a 0.0001 probability (gap open penalty) of going from a match state to either an insertion or a deletion, e.g., gapped, state. Further, when in either a gapped insertion or gapped deletion state there may be only a 0.1 probability (gap extension or continuation penalty) of staying in that gapped state, while there is a 0.9 probability of returning to a match state. It is to be noted that according to this model, all of the probabilities in to or out of a given state should sum to one. Particularly, the processing of the matrix **30** revolves around calculating the transition probabilities, accounting for the various gap open or gap continuation penalties and a final sum is calculated.

[0552] Hence, these calculated state transition probabilities are derived mainly from the directly adjoining cells in the matrix **30**, such as from the cells that are immediately to the left of, the top of, and diagonally up and left of that given cell presently being calculated, as seen in FIG. **21**. Additionally, the state transition probabilities may in part be derived from the "Phred" quality score that accompanies each read base. These transition probabilities, therefore, are useful in computing the M, I, and D state values for that particular cell, and likewise for any associated new cell being calculated. It is to be noted that as described herein, the gap open and gap continuation penalties may be fixed values, however, in various instances, the gap open and gap continuation penalties may be variable and therefore programmable within the system, albeit by employing additional hardware resources dedicated to determining such variable transition probability calculations. Such instances may be useful where greater accuracy is desired. Nevertheless, when such values are assumed to be constant, smaller resource usage and/or chip size may be achieved, leading to greater processing speed, as explained below.

[0553] Accordingly, there is a multiplicity of calculations and/or other mathematical computations, such as multiplications and/or additions, which are involved in deriving each new M, I, and D state value (see FIG. **22**). In such an instance, such as for calculating maximum throughput, the primitive mathematical computations involved in each M, I, and D transition state calculation may be pipelined. Such pipelining may be configured in a way that the corresponding clock frequencies are high, but where the pipeline depth may be non-trivial. Further, such a pipeline may be configured to have a finite depth, and in such instances it may take more than one clock cycle to complete the operations.

[0554] For instance, these computations may be run at high speeds inside the processor **7**, such as at about 300 MHz. This may be achieved such as by pipelining the FPGA or ASIC heavily with registers so little mathematical computation occurs between each flip-flop. This pipeline structure results in multiple cycles of latency in going from the input of the match state to the output, but given the reverse diagonal computing structure, set forth in FIG. **20** above, these latencies may be hidden over the entire HMM matrix **30**, such as where each cell represents one clock cycle.

[0555] Accordingly, the number of M, I, and D state calculations may be limited. In such an instance, the processing engine **13** may be configured in such a manner that a grouping, e.g., swath **35**, of cells in a number of rows of the matrix **30** may be processed as a group (such as in a down-and-left-diagonal fashion as illustrated by the arrow in FIG. **8**) before proceeding to the processing of a second swath below, e.g., where the second swath contains the same number of cells in rows to be processed as the first. In a manner such as this, a hardware implementation of an accelerator **8**, as described herein, may be adapted so as to make the overall system more efficient, as described above.

[0556] A further efficiency may be achieved in instances such as this by limiting state storage requirements to a single row of M, I, and D state values, such as at the bottom edge of the grouping **35** (see row **35** of FIG. **29**). Hence, when starting the processing from one swath **35*a*** to the next **35*b***, e.g., grouping of rows, (below the current swath or grouping), the M, I, and D state values that were stored in the state memory for the previous swath **35*a*** may be used as the edge and/or initial conditions for the cells in the top row of the next swath, e.g., grouping, of cells **35*b*** to be processed. For instance, in an exemplary embodiment, the swath **35*a*** is configured to be 10 cells in length, consequently, the next grouping of cells to be processed **35*b*** will include the next 10 rows of virtual cells in the matrix, such as where the values set for the final row of the first swath **35*a*** being processed set the edge for the values of the next swath **35*b*** of cells to be processed. It is to be noted that the swath length can be any suitable length, such as 2 or 4 or 5 or 10 or 15 or 20 or 25 or 50 cells in length or more.

[0557] Particularly, FIG. **22** sets forth an exemplary computational structure for performing the various state processing calculations herein described. More particularly, FIG. **22** sets forth three dedicated logic blocks **17** of the processing engine **13** for computing the state computations involved in generating each M, I, and D state value for each particular cell, or grouping of cells, being processed in the HMM matrix **30**. As can be seen with respect to FIG. **10**, the match state computation **15*a*** is more involved than either of the insert **15*b*** or deletion **15*c*** computations, this is because in calculating the match state **15*a*** of the present cell being processed, all of the previous match, insert, and delete states of the adjoining cells along with various "Priors" data are included in the present match computation (see FIGS. **21** and **22**), whereas only the match and either the insert and delete states are included in their respective calculations. Hence, as can be seen with respect to FIG. **22**, in calculating a match state, three state multipliers, as well as two adders, and a final multiplier, which accounts for the Prior, e.g. Phred, data are included. However, for calculating the I or D state, only two multipliers and one adder are included. It is noted that in hardware, multipliers are more resource intensive than adders.

[0558] Accordingly, to various extents, the M, I, and D state values for processing each new cell in the HMM matrix **30** uses the knowledge or pre-computation of the following values, such as the "previous" M, I, and D state values from left, above, and/or diagonally left and above of the currently-being-computed cell in the HMM matrix. Additionally, such values representing the prior information, or "Priors", may at least in part be based on the "Phred" quality score, and whether the read base and the reference base at a given cell in the matrix **30** match or are different. Such information is particularly useful when determining a match state. Specifically, as can be seen with respect to FIG. **22**, in such instances, there are basically seven "transition probabilities" (M-to-M, I-to-M, D-to-M, I-to-I, M-to-I, D-to-D, and M-to-D) that indicate and/or estimate the probability of seeing a gap open, e.g., of seeing a transition from a match state to an insert or delete state; seeing a gap close; e.g., going from an insert or delete state back to a match state; and seeing the next state continuing in the same state as the previous state, e.g., Match-to-Match, Insert-to-Insert, Delete-to-Delete.

[0559] The state values (e.g., in any cell to be processed in the HMM matrix **30**), Priors, and transition probabilities are all values in the range of [0,1]. Additionally, there are also known starting conditions for cells that are on the left or top edge of the HMM matrix **30**. As can be seen from the logic **15***a* of FIG. **22**, there are four multiplication and two addition computations that may be employed in the particular M state calculation being determined for any given cell being processed. Likewise, as can be seen from the logic of **15***b* and **15***c* there are two multiplications and one addition involved for each I state and each D state calculation, respectively. Collectively, along with the priors multiplier this sums to a total of eight multiplications and four addition operations for the M, I, and D state calculations associated with each single cell in the HMM matrix **8** to be processed.

[0560] As can be seen with respect to FIG. **29**, the final sum output, e.g., row **34**, of the computation of the matrix **30**, e.g., for a single job **20** of comparing one read to one or two haplotypes, is the summation of the final M and I states across the entire bottom row **34** of the matrix **30**, which is the final sum value that is output from the HMM accelerator **8** and delivered to the CPU **1000**. This final summed value represents how well the read matches the haplotype(s). The value is a probability, e.g., of less than one, for a single job **20***a* that may then be compared to the output resulting from another job **20***b* such as form the same active region **500**. It is noted that there are on the order of 20 trillion HMM cells to evaluate in a "typical" human genome at 30× coverage, where these 20 trillion HMM cells are spread across about 1 to 2 billion HMM matrices **30** of all associated HMM jobs **20**.

[0561] The results of such calculations may then be compared one against the other so as to determine, in a more precise manner, how the genetic sequence of a subject differs, e.g., on a base by base comparison, from that of one or more reference genomes. For the final sum calculation, the adders already employed for calculating the M, I, and/or D states of the individual cells may be re-deployed so as to compute the final sum value, such as by including a mux into a selection of the re-deployed adders thereby including one last additional row, e.g., with respect to calculation time, to the matrix so as to calculate this final sum, which if the read length is 100 bases amounts to about a 1% overhead. In alternative embodiments, dedicated hardware resources can be used for performing such calculations. In various instances, the logic for the adders for the M and D state calculations may be deployed for calculating the final sum, which D state adder may be efficiently deployed since it is not otherwise being used in the final processing leading to the summing values.

[0562] In certain instances, these calculations and relevant processes may be configured so as to correspond to the output of a given sequencing platform, such as including an ensemble of sequencers, which as a collective may be capable of outputting (on average) a new human genome at 30× coverage every 28 minutes (though they come out of the sequencer ensemble in groups of about 150 genomes every three days). In such an instance, when the present mapping, aligning, and variant calling operations are configured to fit within such a sequencing platform of processing technologies, a portion of the 28 minutes (e.g., about 10 minutes) it takes for the sequencing cluster to sequence a genome, may be used by a suitably configured mapper and/or aligner, as herein described, so as to take the FASTQ file results from the sequencer and perform the steps of mapping and/or aligning the genome, e.g., post-sequencer processing. That leaves about 18 minutes of the sequencing time period for performing the variant calling step, of which the HMM operation is the main computational component, such as prior to the nucleotide sequencer sequencing the next genome, such as over the next 28 minutes. Accordingly, in such instances, 18 minutes may be budgeted to computing the 20 trillion HMM cells that need to be processed in accordance with the processing of a genome, such as where each of the HMM cells to be processed includes about twelve mathematical operations (e.g., eight multiplications and/or four addition operations). Such a throughput allows for the following computational dynamics (20 trillion HMM cells)×(12 math ops per cell)/(18 minutes×60 seconds/minute), which is about 222 billion operations per second of sustained throughput.

[0563] Assuming there will be around 10% overhead in loading data into the HMM accelerator, reading results from the accelerator, and in general control of the overhead, one can derive that about 65~70 HMM cells need to be computed each clock cycle. Hence, in various instances, the system may be configured to take 18 minutes for computing the 20 trillion HMM cells so as to achieve a throughput of about 222 billion operations per second. In such an instance, the HMM accelerator can be run at a frequency of 300 MHz so as to achieve this throughput. If more computations are needed to be performed, the computing resources and/or clock frequencies, e.g., higher frequencies, may be configured to accommodate the increased computations

[0564] In these embodiments, the HMM matrix **30**, set forth in FIGS. **20** and **29**, and its resultant computations may not be particularly latency-sensitive. For instance, even with just one HMM cell computed per clock cycle at 300 MHz, the average HMM job (computing all the M, I, and D states and final sum value) will complete in about 60 microseconds. Further, if the memory is limited with respect to a given chip configuration, the fixed cost of the input memories (for read and haplotype data) and the M, I, D state memories may be amortized over multiple HMM cell computation engines **13** per HMM job (per HMM matrix computation **20**).

[0565] FIG. **23** sets forth the logic blocks **17** of the processing engine of FIG. **22** including exemplary M, I, and D state update circuits that present a simplification of the circuit provided in FIG. **22**. The system may be configured so as to not be memory-limited, so a single HMM engine instance **13** (e.g., that computes all of the single cells in the HMM matrix **30** at a rate of one cell per clock cycle, on average, plus overheads) may be replicated multiple times (at least 65~70 times to make the throughput efficient, as described above). Nevertheless, to minimize the size of the hardware, e.g., the size of the chip **2** and/or its associated resource usage, and/or in a further effort to include as many HMM engine instances **13** on the chip **2** as desirable and/or possible, simplifications may be made with regard to the logic blocks **15***a*'-*c*' of the processing instance **13** for computing one or more of the transition probabilities to be calculated.

[0566] In particular, it may be assumed that the gap open penalty (GOP) and gap continuation penalty (GCP), as described above, such as for inserts and deletes are the same and are known prior to chip configuration. This simplification implies that the I-to-M and D-to-M transition probabilities are identical, e.g., see FIG. **26**. In such an instance, one or more of the multipliers, e.g., set forth in FIG. **22**, may be eliminated, such as by pre-adding I and D states before multiplying by a common Indel-to-M transition probability. For instance, in various instances, if the I and D state calculations are assumed to be the same, then the state calculations per cell can be simplified as presented in FIG. **26**. Particularly, if the I and D state values are the same, then the I state and the D state may be added and then that sum may be multiplied by a single value, thereby saving a multiply. This may be done because, as seen with respect to FIG. **25**, the gap continuation and/or close penalties for the I and D states are the same. However, as indicated above, the system can be configured to calculate different values for both the I and D transition state probabilities, and in such an instance, this simplification would not be employed.

[0567] Additionally, in a further simplification, rather than dedicate chip resources configured specifically to perform the final sum operation at the bottom of the HMM matrix, e.g., see row **34** of FIG. **29**, the present HMM accelerator **8** may be configured so as to effectively append one or more additional rows to the HMM matrix **30**, with respect to computational time, e.g., overhead, it takes to perform the calculation, and may also be configured to "borrow" one or more adders from the M-state **15***a* and D-state **15***c* computation logic such as by MUXing in the final sum values to the existing adders as needed, so as to perform the actual final summing calculation. In such an instance, the final logic, including the M logic **15***a*, I logic **15***b*, and D logic **15***c* blocks, which blocks together form part of the HMM MID instance **17**, may include 7 multipliers and 4 adders along with the various MUXing involved.

[0568] Accordingly, FIG. **23** sets forth the M, I, and D state update circuits **15***a*', **15***b*', and **15***c*' including the effects of simplifying assumptions related to transition probabilities, as well as the effect of sharing various M, I, and/or D resources, e.g., adder resources, for the final sum operations. A delay block may also be added to the M-state path in the M-state computation block, as shown in FIG. **22**. This delay may be added to compensate for delays in the actual hardware implementations of the multiply and addition operations, and/or to simplify the control logic, e.g., **15**.

[0569] As shown in FIGS. **22** and **23**, these respective multipliers and/or adders may be floating point multipliers and adders. However, in various instances, as can be seen with respect to FIG. **19**, a log domain configuration may be implemented where in such configuration all of the multiplies turn into adds. FIG. **24** presents what log domain calculation would look like if all the multipliers turned into adders, e.g., **15***a*", **15***b*", and **15***c*", such as occurs when employing a log domain computational configuration. Particularly, all of the multiplier logic turns into an adder, but the adder itself turns into or otherwise includes a function where the function such as: $f(a,b)=\max(a,b)-\log_2(1+2\hat{}(-[a-b])$, such as where the log portion of the equation may be maintained within a LUT whose depth and physical size is determined by the precision required.

[0570] Given the typical read and haplotype sequence lengths as well as the values typically seen for read quality (Phred) scores and for the related transition probabilities, the dynamic range requirements on the internal HMM state values may be quite severe. For instance, when implementing the HMM module in software, various of the HMM jobs **20** may result in underruns, such as when implemented on single-precision (32-bit) floating-point state values. This implies a dynamic range that is greater than 80 powers of 10, thereby requiring the variant call software to bump up to double-precision (64-bit) floating point state values. However, full 64-bit double-precision floating-point representation may, in various instances, have some negative implications, such as if compact, high-speed hardware is to be implemented, both storage and compute pipeline resource requirements will need to be increased, thereby occupying greater chip space, and/or slowing timing. In such instances, a fixed-point-only linear-domain number representation may be implemented. Nevertheless, the dynamic range demands on the state values, in this embodiment, make the bit widths involved in certain circumstances less than desirable. Accordingly, in such instances, fixed-point-only log-domain number representation may be implemented, as described herein.

[0571] In such a scheme, as can be seen with respect to FIG. **24**, instead of representing the actual state value in memory and computations, the −log-base-2 of the number may be represented. This may have several advantages, including employing multiply operations in linear space that translate into add operations in log space; and/or this log domain representation of numbers inherently supports wider dynamic range with only small increases in the number of integer bits. These log-domain M, I, D state update calculations are set forth in FIG. **24**.

[0572] As can be seen when comparing the logic **17** configuration of FIG. **24** with that of FIG. **22**, the multiply operations go away in the log-domain. Rather, they are replaced by add operations, and the add operations are morphed into a function that can be expressed as a max operation followed by a correction factor addition, e.g., via a LUT, where the correction factor is a function of the difference between the two values being summed in the log-domain. Such a correction factor can be either computed or generated from the look-up-table. Whether a correction factor computation or look-up-table implementation is more efficient to be used depends on the required precision (bit width) on the difference between the sum values. In particular instances, therefore, the number of log-domain bits for state representation can be in the neighborhood of 8 to 12 integer bits plus 6 to 24 fractional bits, depending on the level of quality desired for any given implementation. This implies somewhere between 14 and 36 bits total for log-domain state value representation. Further, it has been determined that there are log-domain fixed-point representations that can provide acceptable quality and acceptable hardware size and speed.

[0573] In various instances, there are three main utilizations of RAM (or RAM-like) storage within each HMM engine instance **13**, which includes the haplotype sequence data storage **16**, read sequence data storage **18**, and M, I, D state storage at the bottom edge of the region (or swath), e.g., via a scratch pad memory. Particularly, the haplotype sequence data, such as received by the system **1** from the CPU **1000**, or a suitably configured sequencer coupled therewith, may contain a 4-bit indicator by which each particular base in the haplotype may be represented, as described above with respect to FIG. **17**. For instance, in various embodiments, a suitable haplotype length for use in the present system may be up to 1008 bases, more or less, dependent on the system configuration. In addition to the haplotype sequence, there are a 32-bit control word and 32-bit haplotype ID that may be stored in the same memory **16**. Accordingly, together, this represents a 128 word×32 bits/word HMEM memory **16**, and the organization for each block of haplotype memory is given in FIG. **17**.

[0574] For throughput reasons, and to better utilize the PCIe Bus connection **5** to the microchip **7**, in various instances, the hardware may be configured to allow one, or two, or more haplotypes to be associated with a given read in a given HMM job **20**. Additionally, as indicated, a ping-pong buffer may be set up to give various software implemented functions the ability to write new HMM job data **20***b*, while a current job **20***a* is still being worked on by a given engine instance **13**. Taken together, this means that there may be four blocks of 128×32 memory associated with haplotype storage, e.g., HMEM **16**, and these may be joined together in a single 512×32 two-port memory (one port for write, one port for read, e.g., with separate clocks for write and read ports), as shown in FIG. **17**.

[0575] Likewise, in certain instances, the read sequence data may contain a 2-bit indicator for representing what each base in the read is supposed to be, a 6-bit read quality score (Phred value) per read base, and a 6-bit gap open penalty (GOP) value (also in Phred-like domain). Together these represent 14-bits per read base. Hence, as can be seen with respect to FIG. **18**, the HMM accelerator **8** may be configured such that information associated with two read bases (e.g., 28-bits total, per above) may be packed into a single 32-bit word. Additionally, a 32-bit control word and a 32-bit read ID may be stored in the same memory **18** as the read sequence data. This all may be packed into a 512 word×32-bits/word RMEM memory **18**, thereby indicating that in certain embodiments, the read sequence length may be about 1020 in length more or less.

[0576] In these instances, one read sequence is typically processed for each HMM job **20**, which as indicated may include a comparison against two haplotype sequences. And like above for the haplotype memory, a ping-pong structure may also be used in the read sequence memory **18** to allow various software implemented functions the ability to write new HMM job information **20***b* while a current job **20***a* is still being processed by the HMM engine instance **13**. Hence, a read sequence storage requirement may be for a single 1024×32 two-port memory (such as one port for write, one port for read, and/or separate clocks for write and read ports).

[0577] Particularly, as described above, in various instances, the architecture employed by the system **1** is configured such that in determining whether a given base in a sequenced sample genome matches that of a corresponding base in one or more reference genomes, a virtual matrix **30** is formed, wherein the reference genome is theoretically set across a horizontal axis, while the sequenced reads, representing the sample genome, is theoretically set in descending fashion down the vertical axis. Consequently, in performing an HMM calculation, the HMM processing engine **13**, as herein described, is configured to traverse this virtual HMM matrix **30**. Such processing can be depicted as in FIG. **20**, as a swath **35** moving diagonally down and across the virtual array performing the various HMM calculations for each cell of the virtual array, as seen in FIG. **21**.

[0578] More particularly, this theoretical traversal involves processing a first grouping of rows of cells **35***a* from the matrix **30** in its entirety, such as for all haplotype and read bases within the grouping, before proceeding down to the next grouping of rows **35***b* (e.g., the next group of read bases). In such an instance, the M, I, and D state values for the first grouping are stored at the bottom edge of that initial grouping of rows so that these M, I, and D state values can then be used to feed the top row of the next grouping (swath) down in the matrix **30**. In various instances, the system **1** may be configured to allow up to 1008 length haplotypes and/or reads in the HMM accelerator **8**, and since the numerical representation employs W-bits for each state, this implies a 1008 word×W-bit memory for M, I, and D state storage.

[0579] Accordingly, as indicated, such memory could be either a single-port or double-port memory. Additionally, a cluster-level, scratch pad memory, e.g., for storing the results of the swath boundary, may also be provided. For instance, in accordance with the disclosure above, the memories discussed already are configured for a per-engine-instance **13** basis. In particular HMM implementations, multiple engine instances **13***a*-$_{(n+1)}$ may be grouped into a cluster **11** that is serviced by a single connection, e.g., PCIe bus **5**, to the PCIe interface **4** and DMA **3** via CentCom **9**. Multiple clusters **11***a*-$_{(n+1)}$ can be instantiated so as to more efficiently utilize PCIe bandwidth using the existing CentCom **9** functionality.

[0580] Hence, in a typical configuration, somewhere between 16 and 64 engines **13**$_m$ are instantiated within a cluster **11***n*, and one to four clusters might be instantiated in a typical FPGA/ASIC implementation of the HMM **8** (e.g., depending on whether it is a dedicated HMM FPGA image or whether the HMM has to share FPGA real estate with the sequencer/mapper/aligner and/or other modules, as herein disclosed). In particular instances, there may be a small amount of memory used at the cluster-level 11 in the HMM hardware. This memory may be used as an elastic First In First Out ("FIFO") to capture output data from the HMM engine instances **13** in the cluster and pass it on to CentCom

9 for further transmittal back to the software of the CPU **1000** via the DMA **3** and PCIe **4**. In theory, this FIFO could be very small (on the order of two 32-bit words), as data are typically passed on to CentCom **9** almost immediately after arriving in the FIFO. However, to absorb potential disrupts in the output data path, the size of this FIFO may be made parametrizable. In particular instances, the FIFO may be used with a depth of 512 words. Thus, the cluster-level storage requirements may be a single 512×32 two-port memory (separate read and write ports, same clock domain).

[0581] FIG. **25** sets forth the various HMM state transitions **17***b* depicting the relationship between Gap Open Penalties (GOP), Gap Close Penalties (GCP), and transition probabilities involved in determining whether and how well a given read sequence matches a particular haplotype sequence. In performing such an analysis, the HMM engine **13** includes at least three logic blocks **17***b*, such as a logic block for determining a match state **15***a*, a logic block for determining an insert state **15***b*, and a logic block for determining a delete state **15***c*. These M, I, and D state calculation logic **17** when appropriately configured function efficiently to avoid high-bandwidth bottlenecks, such as of the HMM computational flow. However, once the M, I, D core computation architecture is determined, other system enhancements may also be configured and implemented so as to avoid the development of other bottlenecks within the system.

[0582] Particularly, the system **1** may be configured so as to maximize the process of efficiently feeding information from the computing core **1000** to the variant caller module **2** and back again, so as not to produce other bottlenecks that would limit overall throughput. One such block that feeds the HMM core M, I, D state computation logic **17** is the transition probabilities and priors calculation block. For instance, as can be seen with respect to FIG. **22**, each clock cycle employs the presentation of seven transition probabilities and one Prior at the input to the M, I, D state computation block **15***a*. However, after the simplifications that result in the architecture of FIG. **23**, only four unique transition probabilities and one Prior are employed for each clock cycle at the input of the M, I, D state computation block. Accordingly, in various instances, these calculations may be simplified and the resulting values generated. Thus, increasing throughput, efficiency, and reducing the possibility of a bottleneck forming at this stage in the process.

[0583] Additionally, as described above, the Priors are values generated via the read quality, e.g., Phred score, of the particular base being investigated and whether, or not, that base matches the hypothesis haplotype base for the current cell being evaluated in the virtual HMM matrix **30**. The relationship can be described via the equations bellow: First, the read Phred in question may be expressed as a probability=10^(−(read Phred/10)). Then the Prior can be computed based on whether the read base matches the hypothesis haplotype base: If the read base and hypothesis haplotype base match: Prior=1−read Phred expressed as a probability. Otherwise: Prior=(read Phred expressed as probability)/3. The divide-by-three operation in this last equation reflects the fact that there are only four possible bases (A, C, G, T). Hence, if the read and haplotype base did not match, then it must be one of the three remaining possible bases that does match, and each of the three possibilities is modeled as being equally likely.

[0584] The per-read-base Phred scores are delivered to the HMM hardware accelerator **8** as 6-bit values. The equations to derive the Priors, then, have 64 possible outcomes for the "match" case and an additional 64 possible outcomes for the "don't match" case. This may be efficiently implemented in the hardware as a 128 word look-up-table, where the address into the look-up-table is a 7-bit quantity formed by concatenating the Phred value with a single bit that indicates whether, or not, the read base matches the hypothesis haplotype base.

[0585] Further, with respect to determining the match to insert and/or match to delete probabilities, in various implementations of the architecture for the HMM hardware accelerator **8**, separate gap open penalties (GOP) can be specified for the Match-to-Insert state transition, and the Match-to-Delete state transition, as indicated above. This equates to the M2I and M2D values in the state transition diagram of FIG. **25** being different. As the GOP values are delivered to the HMM hardware accelerator **8** as 6-bit Phred-like values, the gap open transition probabilities can be computed in accordance with the following equations: M2I transition probability=10^(−(read GOP(I)/10)) and M2D transition probability=10^(−(read GOP(D)/10)). Similar to the Priors derivation in hardware, a simple 64 word look-up-table can be used to derive the M2I and M2D values. If GOP(I) and GOP(D) are inputted to the HMM hardware **8** as potentially different values, then two such look-up-tables (or one resource-shared look-up-table, potentially clocked at twice the frequency of the rest of the circuit) may be utilized.

[0586] Furthermore, with respect to determining match to match transition probabilities, in various instances, the match-to-match transition probability may be calculated as: M2M transition probability=1−(M2I transition probability+M2D transition probability). If the M2I and M2D transition probabilities can be configured to be less than or equal to a value of ½, then in various embodiments the equation above can be implemented in hardware in a manner so as to increase overall efficiency and throughput, such as by reworking the equation to be: M2M transition probability=(0.5−M2I transition probability)+(0.5−M2D transition probability). This rewriting of the equation allows M2M to be derived using two 64 element look-up-tables followed by an adder, where the look-up-tables store the results.

[0587] Further still, with respect to determining the Insert to Insert and/or Delete to Delete transition probabilities, the I2I and D2D transition probabilities are functions of the gap continuation probability (GCP) values inputted to the HMM hardware accelerator **8**. In various instances, these GCP values may be 6-bit Phred-like values given on a per-read-base basis. The I2I and D2D values may then be derived as shown: I2I transition probability=10^(−(read GCP(I)/10)), and D2D transition probability=10^(−(read GCP(D)/10)). Similar to some of the other transition probabilities discussed above, the I2I and D2D values may be efficiently implemented in hardware, and may include two look-up-tables (or one resource-shared look-up-table), such as having the same form and contents as the Match-to-Indel look-up-tables discussed previously. That is, each look-up-table may have 64 words.

[0588] Additionally, with respect to determining the Inset and/or Delete to Match probabilities, the I2M and D2M transition probabilities are functions of the gap continuation probability (GCP) values and may be computed as: I2M transition probability=1−I2I transition probability, and D2M transition probability=1−D2D transition probability, where the I2I and D2D transition probabilities may be derived as discussed above. A simple subtract operation to implement the equations above may be more expensive in hardware resources than simply implementing another 64 word look-up-table and using two copies of it to implement the I2M and D2M derivations. In such instances, each look-up-table may have 64 words. Of course, in all relevant embodiments, simple or complex subtract operations may be formed with the suitably configured hardware.

[0589] FIG. **26** provides the circuitry **17***a* for a simplified calculation for HMM transition probabilities and Priors, as described above, which supports the general state transition diagram of FIG. **25**. As can be seen with respect to FIG. **23**, in various instances, a simple HMM hardware accelerator architecture **17** is presented, which accelerator may be configured to include separate GOP values for Insert and Delete transitions, and/or there may be separate GCP values for Insert and Delete transitions. In such an instance, the cost of generating the seven unique transition probabilities and one Prior each clock cycle may be configured as set forth below: eight 64 word look-up-tables, one 128 word look-up-table, and one adder.

[0590] Further, in various instances, the hardware **2**, as presented herein, may be configured so as to fit as many HMM engine instances **13** as possible onto the given chip target (such as on an FPGA, sASIC, or ASIC). In such an instance, the cost to implement the transition probabilities and priors generation logic **17***a* can be substantially reduced relative to the costs as provided by the below configurations. Firstly, rather than supporting a more general version of the state transitions, such as set forth in FIG. **25**, e.g., where there may be separate values for GOP(I) and GOP(D), rather, in various instances, it may be assumed that the GOP values for insert and delete transitions are the same for a given base. This results in several simplifications to the hardware, as indicated above and shown with respect to FIG. **27**.

[0591] In such instances, only one 64 word look-up-table may be employed so as to generate a single M2Indel value, replacing both the M2I and M2D transition probability values, whereas two tables are typically employed in the more general case. Likewise, only one 64 word look-up-table may be used to generate the M2M transition probability value, whereas two tables and an add may typically be employed in the general case, as M2M may now be calculated as 1−2×M2Indel.

[0592] Secondly, the assumption may be made that the sequencer-dependent GCP value for both insert and delete are the same AND that this value does not change over the course of an HMM job **20**. This means that: a single Indel2Indel transition probability may be calculated instead of separate I2I and D2D values, using one 64 word look-up-table instead of two tables; and single Indel2Match transition probability may be calculated instead of separate I2M and D2M values, using one 64 word look-up-table instead of two tables.

[0593] Additionally, a further simplifying assumption can be made that assumes the Inset2Insert and Delete2Delete (I2I and D2D) and Insert2Match and Delete2Match (I2M and D2M) values are not only identical between insert and delete transitions, but may be static for the particular HMM job **20**. Thus, the four look-up-tables associated in the more general architecture with I2I, D2D, I2M, and D2M transition probabilities can be eliminated altogether. In various of these instances, the static Indel2Indel and Indel2Match probabilities could be made to be entered via software or via an RTL parameter (and so would be bitstream programmable in an FPGA). In certain instances, these values may be made bitstream-programmable, and in certain instances, a training mode may be implemented employing a training sequence so as to further refine transition probability accuracy for a given sequencer run or genome analysis.

[0594] FIG. **27** sets forth what the new state transition **17***b* diagram may look like when implementing these various simplifying assumptions. Specifically, FIG. **27** sets forth the simplified HMM state transition diagram depicting the relationship between GOP, GCP, and transition probabilities with the simplifications set forth above.

[0595] Likewise, FIG. **28** sets forth the circuitry **17***a,b* for the HMM transition probabilities and priors generation, which supports the simplified state transition diagram of FIG. **27**. As seen with respect to FIG. **28**, a circuit realization of that state transition diagram is provided. Thus, in various instances, for the HMM hardware accelerator **8**, the cost of generating the transition probabilities and one Prior each clock cycle reduces to: Two 64 word look-up-tables, and One 128 word look-up-table.

[0596] As set forth above, the engine control logic **15**/**17** is configured for generating the virtual matrix and/or traversing the matrix so as to reach the edge of the swath, e.g., via high-level engine state machines, where result data may be finally summed, e.g., via final sum control logic **19**, and stored, e.g., via put/get logic. FIG. **29** presents a representation of an exemplary virtual matrix **30** with the hypothesis haplotype sequence index positioned along the horizontal axis and the read sequence index positioned along the vertical axis. Specifically, FIG. **29** illustrates an exemplary method by which such a virtual HMM matrix **30** may be traversed.

[0597] Accordingly, as can be seen with respect to FIG. **29**, in various embodiments, a method for producing and/or traversing an HMM cell matrix **30** is provided. Specifically, FIG. **29** sets forth an example of how the HMM accelerator control logic **15** goes about traversing the virtual cells in the HMM matrix. For instance, assuming for exemplary purposes, a 5 clock cycle latency for each multiply and each add operation, the worst-case latency through the M, I, D state update calculations would be the 20 clock cycles it would take to propagate through the M update calculation, e.g., see FIG. **21**. There are half as many operations in the I and D state update calculations, implying a 10 clock cycle latency for those operations.

[0598] These latency implications of the M, I, and D compute operations can be understood with respect to FIGS. **21** and **22**, which sets forth various examples of the cell-to-cell data dependencies. In such instances, the M and D state information of a given cell feed the D state computations of the cell in the HMM matrix that is immediately to the right (e.g., having the same read base as the given cell, but having the next haplotype base). Likewise, the M and I state information for the given cell feed the I state computations of the cell in the HMM matrix that is immediately below (e.g., having the same haplotype base as the give cell, but having the next read base). So, in particular instances, the M, I, and D states of a given cell feed the D and I state computations of cells in the next diagonal of the HMM cell matrix.

[0599] Similarly, the M, I, and D states of a given cell feed the M state computation of the cell that is to the right one and down one (e.g., having both the next haplotype base AND the next read base). This cell is actually two diagonals away from the cell that feeds it (whereas, the I and D state calculations rely on states from a cell that is one diagonal away). This quality of the I and D state calculations relying on cells one diagonal away, while the M state calculations rely on cells two diagonals away, has a beneficial result for hardware design.

[0600] Particularly, given these configurations, I and D state calculations may be adapted to take half as long (e.g., 10 cycles) as the M state calculations (e.g., 20 cycles). Hence, if M state calculations are started 10 cycles before I and D state calculations for the same cell, then the M, I, and D state computations for a cell in the HMM matrix **30** will all complete at the same time. Additionally, if the matrix **30** is traversed in a diagonal fashion, such as having a swath **35** of about 10 cells each within it (e.g., that spans ten read bases), then: The M and D states produced by a given cell at (hap, rd) coordinates (i, j) can be used by cell (i+1, j) D state calculations as soon as they are all the way through the compute pipeline of the cell at (i, j).

[0601] The M and I states produced by a given cell at (hap, rd) coordinates (i, j) can be used by cell (i, j+1) I state calculations one clock cycle after they are all the way through the compute pipeline of the cell at (i, j). Likewise, the M, I and D states produced by a given cell at (hap, rd) coordinates (i, j) can be used by cell (i+1, j+1) M state calculations one clock cycle after they are all the way through the compute pipeline of the cell at (i, j). Taken together, the above points establish that very little dedicated storage is needed for the M, I, and D states along the diagonal of the swath path that spans the swath length, e.g., of ten reads. In such an instance, just the registers required to delay cell (i, j) M, I, and D state values one clock cycle for use in cell (i+1, j+1) M calculations and cell (i, j+1) I calculations by one clock cycle). Moreover, there is somewhat of a virtuous cycle here as the M state computations for a given cell are begun 10 clock cycles before the I and D state calculations for that same cell, natively outputting the new M, I, and D states for any given cell simultaneously.

[0602] In view of the above, and as can be seen with respect to FIG. **29**, the HMM accelerator control logic **15** may be configured to process the data within each of the cells of the virtual matrix **30** in a manner so as to traverse the matrix. Particularly, in various embodiments, operations start at cell (0, 0), with M state calculations beginning 10 clock cycles before I and D state calculations begin. The next cell to traverse should be cell (1, 0). However, there is a ten cycle latency after the start of I and D calculations before the results from cell (0, 0) will be available. The hardware, therefore, inserts nine "dead" cycles into the compute pipeline. These are shown as the cells with haplotype index less than zero in FIG. **29**.

[0603] After completing the dead cycle that has an effective cell position in the matrix of (−9, −9), the M, I, and D state values for cell (0, 0) are available. These (e.g., the M and D state outputs of cell (0, 0)) may now be used straight away to start the D state computations of cell (0, 1). One clock cycle later, the M, I, and D state values from cell (0, 0) may be used to begin the I state computations of cell (0, 1) and the M state computations of cell (1, 1).

[0604] The next cell to be traversed may be cell (2, 0). However, there is a ten cycle latency after the start of I and D calculations before the results from cell (1, 0) will be available. The hardware, therefore, inserts eight dead cycles into the compute pipeline. These are shown as the cells with haplotype index less than zero, as in FIG. **24** along the same diagonal as cells (1, 0) and (0, 1). After completing the dead cycle that has an effective cell position in the matrix of (−8, −9), the M, I, and D state values for cell (1, 0) are available. These (e.g., the M and D state outputs of cell (1, 0)) are now used straight away to start the D state computations of cell (2, 0).

[0605] One clock cycle later, the M, I, and D state values from cell (1, 0) may be used to begin the I state computations of cell (1, 1) and the M state computations of cell (2, 1). The M and D state values from cell (0, 1) may then be used at that same time to start the D state calculations of cell (1, 1). One clock cycle later, the M, I, and D state values from cell (0, 1) are used to begin the I state computations of cell (0, 2) and the M state computations of cell (1, 2).

[0606] Now, the next cell to traverse may be cell (3, 0). However, there is a ten-cycle latency after the start of I and D calculations before the results from cell (2, 0) will be available. The hardware, therefore, inserts seven dead cycles into the compute pipeline. These are again shown as the cells with haplotype index less than zero in FIG. **24** along the same diagonal as cells (2, 0), (1, 1), and (0, 2). After completing the dead cycle that has an effective cell position in the matrix of (−7, −9), the M, I, and D state values for cell (2, 0) are available. These (e.g., the M and D state outputs of cell (2, 0)) are now used straight away to start the D state computations of cell (3, 0). And, so, computation for another ten cells in the diagonal begins.

[0607] Such processing may continue until the end of the last full diagonal in the swath **35***a*, which, in this example (that has a read length of 35 and haplotype length of 14), will occur after the diagonal that begins with the cell at (hap, rd) coordinates of (13, 0) is completed. After the cell (4, 9) in FIG. **28** is traversed, the next cell to traverse should be cell (13, 1). However, there is a ten-cycle latency after the start of the I and D calculations before the results from cell (12, 1) will be available.

[0608] The hardware may be configured, therefore, to start operations associated with the first cell in the next swath **35***b*, such as at coordinates (0, 10). Following the processing of cell (0, 10), then cell (13, 1) can be traversed. The whole diagonal of cells beginning with cell (13, 1) is then traversed until cell (5, 9) is reached. Likewise, after the cell (5, 9) is traversed, the next cell to traverse should be cell (13, 2). However, as before there may be a ten cycle latency after the start of I and D calculations before the results from cell (12, 2) will be available. Hence, the hardware may be configured to start operations associated with the first cell in the second diagonal of the next swath **35***b*, such as at coordinates (1, 10), followed by cell (0, 11).

[0609] Following the processing of cell (0, 11), the cell (13, 2) can be traversed, in accordance with the methods disclosed above. The whole diagonal **35** of cells beginning with cell (13,2) is then traversed until cell (6, 9) is reached. Additionally, after the cell (6, 9) is traversed, the next cell to be traversed should be cell (13, 3). However, here again there may be a ten-cycle latency period after the start of the I and D calculations before the results from cell (12, 3) will be available. The hardware, therefore, may be configured to start operations associated with the first cell in the third diagonal of the next swath **35***c*, such as at coordinates (2, 10), followed by cells (1, 11) and (0, 12), and likewise.

[0610] This continues as indicated, in accordance with the above until the last cell in the first swath **35***a* (the cell at (hap, rd) coordinates (13, 9)) is traversed, at which point the logic can be fully dedicated to traversing diagonals in the second swath **35***b*, starting with the cell at (9, 10). The pattern outlined above repeats for as many swaths of 10 reads as necessary, until the bottom swath **35***c* (those cells in this example that are associated with read bases having index **30**, or greater) is reached.

[0611] In the bottom swath **35**, more dead cells may be inserted, as shown in FIG. **29** as cells with read indices greater than 35 and with haplotype indices greater than 13. Additionally, in the final swath **35***c*, an additional row of cells may effectively be added. These cells are indicated at line **35** in FIG. **29**, and relate to a dedicated clock cycle in each diagonal of the final swath where the final sum operations are occurring. In these cycles, the M and I states of the cell immediately above are added together, and that result is itself summed with a running final sum (that is initialized to zero at the left edge of the HMM matrix **30**).

[0612] Taking the discussion above as context, and in view of FIG. **29**, it is possible to see that, for this example of read length of 35 and haplotype length of 14, there are 102 dead cycles, 14 cycles associated with final sum operations, and 20 cycles of pipeline latency, for a total of 102+14+20=146 cycles of overhead. It can also be seen that, for any HMM job **20** with a read length greater than 10, the dead cycles in the upper left corner of FIG. **29** are independent of read length. It can also be seen that the dead cycles at the bottom and bottom right portion of FIG. **29** are dependent on read length, with fewest dead cycles for reads having mod(read length, 10)=9 and most dead cycles for mod(read length, 10)=0. It can further be seen that the overhead cycles become smaller as a total percentage of HMM matrix **30** evaluation cycles as the haplotype lengths increase (bigger matrix, partially fixed number of overhead cycles) or as the read lengths increase (note: this refers to the percentage of overhead associated with the final sum row in the matrix being reduced as read length—row-count—increases). Using such histogram data from representative whole human genome runs, it has been determined that traversing the HMM matrix in the manner described above results in less than 10% overhead for the whole genome processing.

[0613] Further methods may be employed to reduce the amount of overhead cycles including: Having dedicated logic for the final sum operations rather than sharing adders with the M and D state calculation logic. This eliminates one row of the HMM matrix **30**. Using dead cycles to begin HMM matrix operations for the next HMM job in the queue.

[0614] Each grouping of ten rows of the HMM matrix **30** constitutes a "swath" **35** in the HMM accelerator function. It is noted that the length of the swath may be increased or decreased so as to meet the efficiency and/or throughput demands of the system. Hence, the swatch length may be about five rows or less to about fifty rows or more, such as about ten rows to about forty-five rows, for instance, about fifteen or about twenty rows to about forty rows or about thirty five rows, including about twenty five rows to about thirty rows of cells in length.

[0615] With the exceptions noted in the section, above, related to harvesting cycles that would otherwise be dead cycles at the right edge of the matrix of FIG. **29**, the HMM matrix may be processed one swath at a time. As can be seen with respect to FIG. **29**, the states of the cells in the bottom row of each swath **35***a* feed the state computation logic in the top row of the next swath **35***b*. Consequently, there may be a need to store (put) and retrieve (get) the state information for those cells in the bottom row, or edge, of each swath.

[0616] The logic to do this may include one or more of the following: when the M, I, and D state computations for a cell in the HMM matrix **30** complete for a cell with mod(read index, 10)=9, save the result to the M, I, D state storage memory. When M and I state computations (e.g., where D state computations do not require information from cells above them in the matrix) for a cell in the HMM matrix **30** begin for a cell with mod(read index, 10)=0, retrieve the previously saved M, I, and D state information from the appropriate place in the M, I, D state storage memory. Note in these instances that M, I, and D state values that feed row 0 (the top row) M and I state calculations in the HMM matrix **30** are simply a predetermined constant value and do not need to be recalled from memory, as is true for the M and D state values that feed column 0 (the left column) D state calculations.

[0617] As noted above, the HMM accelerator may or may not include a dedicated summing resource in the HMM hardware accelerator such that exist simply for the purpose of the final sum operations. However, in particular instances, as described herein, an additional row may be added to the bottom of the HMM matrix **30**, and the clock cycles associated with this extra row may be used for final summing operations. For instance, the sum itself may be achieved by borrowing (e.g., as per FIG. **26**) an adder from the M state computation logic to do the M+I operation, and further by borrowing an adder from the D state computation logic to add the newly formed M+I sum to the running final sum accumulation value. In such an instance, the control logic to activate the final sum operation may kick in whenever the read index that guides the HMM traversing operation is equal to the length of the inputted read sequence for the job. These operations can be seen at line **34** toward the bottom of the sample HMM matrix **30** of FIG. **29**.

[0618] Particularly, once the genetic data has been generated and/or processed, e.g., in one or more primary and/or secondary processing protocols, such as by being mapped, aligned, and/or sorted, such as to produce one or more variant call files, for instance, to determine how the genetic sequence data from a subject differs from one or more reference sequences, a further aspect of the disclosure may be directed to performing one or more other analytical functions on the generated and/or processed genetic data such as for further, e.g., tertiary, processing. For example, the system may be configured for further processing of the generated and/or secondarily processed data, such as by running it through one or more tertiary processing pipelines, such as one or more of a genome pipeline, an epigenome pipeline, metagenome pipeline, joint genotyping, a MuTect2 pipeline, or other tertiary processing pipeline, such as by the devices and methods disclosed herein. For instance, in various instances, an additional layer of processing may be provided, such as for disease diagnostics, therapeutic treatment, and/or prophylactic prevention, such as including NIPT, NICU, Cancer, LDT, AgBio, and other such disease diagnostics, prophylaxis, and/or treatments employing the data generated by one or more of the present primary and/or secondary and/or tertiary pipelines. Hence, the devices and methods herein disclosed may be used to generate genetic sequence data, which data may then be used to generate one or more variant call files and/or other associated data that may further be subject to the execution of other tertiary processing pipelines in accordance with the devices and methods disclosed herein, such as for particular and/or general disease diagnostics as well as for prophylactic and/or therapeutic treatment and/or developmental modalities.

[0619] Accordingly, as set forth herein above, in various aspects, this present disclosure is directed to systems, apparatuses, and methods for implementing genomics and/or bioinformatic protocols such as, in various instances, for performing one or more functions for analyzing genetic data on an integrated circuit, such as implemented in a hardware processing platform. For example, in one aspect, a bioinformatics system is provided, wherein the system may involve the performance of various bioanalytical functions that have been optimized so as to be performed faster and/or with increased accuracy in a hardware implementation. Accordingly, in various instances, the methods and systems herein described may include the performance of one or more algorithms for executing these functions, wherein the algorithms may be implemented in a hardware solution, such as where the algorithm has been optimized so as to be implemented by an integrated circuit formed of one or more hardwired digital logic circuits. In such an instance, the hardwired digital logic circuits may be interconnected, such as by one or a plurality of physical electrical interconnects, and may be arranged to function as one or more processing engines. In various instances, a plurality of hardwired digital logic circuits are provided, which hardwired digital logic circuits are configured as a set of processing engines, wherein each processing engine is capable of performing one or more steps in the bioinformatics genetic analysis protocol.

[0620] More particularly, in one instance, a system for executing a sequence analysis pipeline such as on genetic sequence data is provided. The system may include one or more of an electronic data source, a memory, and an integrated circuit. For instance, in one embodiment, an electronic data source is included, where in the electronic data source may be configured for providing one or more digital signals, such as a digital signal representing one or more reads of genetic data, for example, where each read of genomic data includes a sequence of nucleotides. Further, the memory may be configured for storing one or more genetic reference sequences, and may further be configured for storing an index, such as an index of the one or more genetic reference sequences.

[0621] Further still, in various instances, one or more of the plurality of physical electrical interconnects may include an input, such as to the integrated circuit, and may further be connected with the electronic data source, so as to be able to receive the one or more reads of genomic data. In various embodiments, the hardwired digital logic circuits may be arranged as a set of processing engines, such as where each processing engine is formed of a subset of the hardwired digital logic circuits, and is configured so as to perform one or more steps in the sequence analysis pipeline, such as on digitized genetic data, e.g., on the plurality of reads of genomic data. In such instances, each subset of the hardwired digital logic circuits may be in a wired configuration so as to perform the one or more steps in the sequence analysis pipeline, such as where the one or more steps may include performing one or more of: a base calling and/or error correction operation, such as on the digitized genetic data, and/or may include one or more of performing a mapping, an alignment, and/or a sorting function on the genetic data. In certain instances, the pipeline may include performing one or more of a realignment, a deduplication, a base quality score recalibration, a reduction and/or compression, and/or a decompression on the digitized genetic data. In certain instances the pipeline may include performing a variant calling operation on the genetic data.

[0622] Accordingly, in various embodiments, the systems, apparatuses, and methods for implementing genomics and/or bioinformatic protocols, as herein described, may involve taking processes that may have typically been performed on software, and embedding those functions into an integrated circuit, such as on a chip **7**, for instance as part of a circuit board **2**, such as where the functions have been optimized to enhance its performance on the chip. Hence, in one embodiment, as can be seen with respect to FIGS. **13** and **41** a chip **2** (**100**) is provided wherein the chip **2** has been designed so as to efficiently perform the functions of the pipeline. In various particular embodiments the chip **7** may be a field programmable gate array (FPGA), an application specific integrated circuit (ASIC), or a structured application specific integrated circuit (sASIC), or the like.

[0623] For instance, the functioning of one or more of these algorithms may be embedded onto a chip, such as into an FPGA or ASIC or structured ASIC chip, and may be optimized so as to perform more efficiently because of their implementation in such hardware. Accordingly, in one embodiment a FPGA chip is provided wherein the chip is capable of being configurable, e.g., its programming may be changed, so as to be more adaptable in meeting a given user's needs with respect to performing the various genomic functions detailed herein. In such an instance, the user can change and/or modify the algorithms employed dependent on the key parameters desired to be emphasized in the overall system, such as to give additional functionality or change out what was first presented on the chip, e.g., such as re-configuring the chip to employ a different algorithm.

[0624] Further, in another embodiment a structured ASIC chip is provided wherein the chip is capable of being configurable such as to a limited extent, e.g., some of its programming may be changed, so as to be more adaptable in meeting a given user's needs with respect to performing the various genomic functions detailed herein. In accordance with another embodiment an ASIC is provided, such as where the FPGA or sASIC is converted to an ASIC chip where its functionality may be locked down into the chip. In such an instance, various parameters, such as various parameters regarding the function of one or more of the algorithms set forth herein, may be user selected, for instance, governing how the various modules are supposed to function, but the way those modules actually function is locked in.

[0625] In various embodiments, as seen with respect to FIGS. **13**, **41**, the chip **7** (**100**) may be part of a circuit board, such as part of an expansion card **2** (**104**), for instance, a peripheral component interconnect (PCI) card, including a PCIe card, which in various embodiments may be associated, such as, communicably coupled, e.g., electrically connected, with an automated sequencer device so as to function part and parcel with the sequencer, such as where the data files, e.g., FASTQ files, generated by the sequencer is transferred directly over to the chip, such as for secondary genomic processing, such as immediately subsequent to the FASTQ file generation and/or primary processing, e.g., immediately after the sequencing function has been performed.

[0626] Accordingly, in certain instances, a PCI **2** card is provided wherein the PCI card may include a chip with a PCIe bus **5**, where the card **2** and/or chip **7** may include one or more of: a configuration manager, such as a configuration control (Cent-Com); a direct memory access engine (e.g., a driver); an API; a client level interface (CLI), a library; a memory, such as a random access memory (RAM) or a dynamic random access memory (DRAM); and/or a chip level interconnect, such as a DDR3. For instance, in various instances a configuration manager may be included wherein the configuration manager is driven, such as by a parameter file. In such an instance the configuration manager may be adapted so as to configure the various modules of the pipeline. In various instances, it may be user editable, and thereby allow a user to determine which modules of the pipeline are going to be used, e.g., from all of them to a subset of less than all of them, such as for a particular dataset, such as a particular set of FASTQ files.

[0627] For example, in various embodiments, the functioning of the pipeline is very configurable such that one or more of the modules, such as structured into the chip, may be run or not run, as desired. Further, each module in use can also be configured so as to run in accordance with one or more preselected parameters, which the user may have control over, such as regarding how the module is going to perform and behave. Hence, there may be two different sets of configuration files, such as one that controls the basic operations of the system as a whole, and may be hidden from the user, and another that is capable of being manipulated by the user, thereby allowing the user to select various of the parameters by which one or more of the subsystems, e.g., modules, of the chip **7** (**100**) and/or PCI card **2** (**104**) will be run.

[0628] Accordingly, various of the above described modules may be hardwired into the chip, or may be external to the chip, but positioned in a coupling relationship therewith, such as on a PCI board **2** (**104**), or they may be located remotely from the chip, such as on a different PCI board, or even on a different server, such as on a server that may be accessed via the cloud **30**. For instance, in certain implementations, one or more of the above described modules may be hardwired onto a chip **7** (**100**) and the chip installed onto the circuit board **2** (**104**) of a stand-alone device **300**, or coupled to a sequencer, whereby the user configures and runs the system directly by themselves according to their own preselected parameters. Alternatively, as indicated herein, one or more of the above described modules may be present on a system that is accessible via the cloud **30**, wherein the directing of the functioning of the pipeline, and/or the modules thereof, may include the user logging on to a server, e.g., a remote server, and transmitting data to and therefrom, and thereby selects which modules to be run on the data set. In certain instances, one or more of the modules may be performed remotely, such as via the cloud accessed server.

[0629] In various instances, in configuring the system, the chip, e.g., the chip **7** (**100**) on an expansion card **2** (**104**), such as a PCI card, may be included in a server **300**, whereby the server runs the various applications of the system. In certain instances, the server **300** may have a terminal connectable there with, whereby a windows interface may be presentable to the user such that the user may select the modules to be run and the parameters by which they are to be run, such as by selecting a box from a menu of boxes. In other instances, however, the parameter file may be a text file detailing categories by module under file names that the user can then edit, so as to select which modules will be run in accordance with which parameters. For instance, in various embodiments, each chip may include all or a selection of the modules, such as one or more of: a base calling, error correcting, a mapping, an alignment, a sorting, a local realignment, a duplicate marking, a recalibration, a variant calling, a compression, and/or a decompression module, from which the user may select which modules will run, when, and to various extents how it will run, without changing the functioning of the underlying algorithms by which the individual modules are operated.

[0630] Additionally, in various instances, a direct memory access (DMA) engine in the chip, and a DMA driver, may be included wherein the DMA driver includes code that runs in the kernel. Accordingly, the DMA driver may be the foundation of the overall operating system. For instance, where the kernel runs in a literal addressing space, layered above that may be a virtual user space. This operating system software, therefore operates in between these layers managing the mapping from the virtual to the physical space. More particularly, the kernel represents the lowest level of code that gives the platform access to the PCI **2**, e.g., PCIe, bus **5**, to which the chip **7** (**100**) is coupled. Accordingly, since, in various embodiments, the chip **7** (**100**) may be configured as an expansion card **2** (**104**) with a PCIe expansion bus **5**, which expansion card **2** may be coupled with various hardware of a device, such as a sequencer, the DMA driver may function so as to communicate with the hardware of the sequencer, and may further be configured for running at the kernel level on the CPU **100**, so as to also communicate with the DMA engine in the chip **7** (**100**), and/or be configured for operating in the virtual user space so as to receive instructions from the user.

[0631] To facilitate this communication within the chip and/or between the chip and one or more cards, every single configurable parameter of a module may be assigned to a register address. In such an instance, the card may have its own address space, which address space may be different from the address space for one or more memories, such as 64 gigabytes of memory, and/or additionally every module may have registers and local memory associated with it, each with its own address space. Accordingly, the driver knows where everything is, all the addresses, and knows how to communicate between the chip **7** (**100**), the PCI card **2** (**104**), and/or the hardware of the server. Further, knowing where all the addresses are and communicating with an API the driver can read the parameter file that a user generates, and can look up for that parameter where the file is actually located in the host computer system and will read and interpret the value in the file and will deliver that value in the right register in the right place in the chip. Hence, the driver may handle delivering the selected parameter instructions, such as with respect to various user selected configurations, and ships that data to the chip via the DMA engine to configure any of its processing functions.

[0632] Particularly, once the genetic data has been generated and/or processed, e.g., in one or more primary and/or secondary processing protocols, such as by being mapped, aligned, and/or sorted, such as to produce one or more variant call files, for instance, to determine how the genetic sequence data from a subject differs from one or more reference sequences, a further aspect of the disclosure may be directed to performing one or more other analytical functions on the generated and/or processed genetic data such as for further, e.g., tertiary, processing. For example, the system, as presented in FIG. **33**, may be configured for further processing of the generated and/or secondarily processed data, such as by running it through one or more tertiary processing pipelines **700**, such as one or more of a genome pipeline, an epigenome pipeline, metagenome pipeline, joint genotyping, a MuTect2 pipeline, or other tertiary processing pipeline, such as by the devices and methods disclosed herein. For instance, in various instances, an additional layer of processing **800** may be provided, such as for disease diagnostics, therapeutic treatment, and/or prophylactic prevention, such as including NIPT, NICU, Cancer, LDT, AgBio, and other such disease diagnostics, prophylaxis, and/or treatments employing the data generated by one or more of the present primary and/or secondary and/or tertiary pipelines.

[0633] Hence, the devices and methods herein disclosed may be used to generate genetic sequence data, which data may then be used to generate one or more variant call files and/or other associated data that may further be subject to the execution of other tertiary processing pipelines in accordance with the devices and methods disclosed herein, such as for particular and/or general disease diagnostics as well as for prophylactic and/or therapeutic treatment and/or developmental modalities.

[0634] Further, in various instances, an API may be included wherein the API is configured so as to include a list of function calls that the user can make, so as to configure and operate the system. For instance, an API may be defined in a header file that describes the functionality and determines how to call a function, such as the parameters that are passed, the inputs and outputs, what comes in, what goes out, and what gets returned. For example, in various embodiments, one or more of the elements of the pipeline may be configurable such as by instructions entered by a user and/or one or more third party applications. These instructions may be communicated to the chip via the API which communicates with the driver, instructing the driver as to which parts of the chip, e.g., which modules are to be activated, when, and in what order, given a preselected parameter configuration.

[0635] As indicated above, the DMA driver runs at the kernel level, and has its own very low level, basic API that provides access to the hardware and functions so as to access applicable registers and modules. On top of this layer is built a virtual layer of service functions, that form the building blocks that are used for a multiplicity of functions that send files down to the kernel and gets results back, and further performs more higher level functions. On top of that layer is an additional layer that uses those service functions, which is the API level that a user will interface with and it functions primarily for configuration, downloading files, and uploading results. Such configuration may include communicating with registers and also performing function calls.

[0636] For example, as described herein above, one function call may be to generate the hash table via the hashing algorithm. Specifically, because in certain embodiments this function may be based on a reference genome, once for every reference genome, the hash tables that are used in the mapper may need to be constructed, based on the reference, there is therefore a function call that performs this function, which function call will accept a file name of where the reference file is stored and it will then generate one or more data files that contain the hash table and the reference. Another function call may be to load the hash table that was generated via the hashing algorithm and transfer that down to the memory on the chip **7**, and/or put it at the right spot where the hardware is expecting them to be. Of course, the reference itself will need to be downloaded onto the chip **7**, as well for the performance of the alignment function, and the configuration manager can perform that function such as by loading everything that needs to be there in order for the modules of the chip **7** to perform their functions into a memory on to the chip or attached to the chip **7**.

[0637] Additionally, the API may be configured to allow the chip **7** to interface with the circuit board of the sequencer, when included therewith, so as to receive the FASTQ sequencing files directly from the sequencer such as immediately once they have been generated and then transfers that information to the configuration manager which then directs that information to the appropriate memory banks in the hardware **7** that makes that information available to the pertinent modules of the hardware so that they can perform their designated functions on that information so as to call bases, map, align, sort, etc. the sample DNA with respect to the reference genome.

[0638] Further still, a client level interface (CLI) may be included wherein the CLI may allow the user to call one or more of these functions directly. In various embodiments, the CLI may be a software application that is adapted to configure the use of the hardware. The CLI, therefore, may be a program that accepts instructions, e.g., arguments, and makes functionality available simply by calling an application program. As indicated above, the CLI can be command line based or GUI (graphical user interface) based. The line based commands happen at a level below the GUI, where the GUI includes a windows based file manager with click on function boxes that delineate which modules will be used and the parameters of their use. For example, in operation, if instructed, the CLI will locate the reference, will determine if a hash table and/or index needs to be generated, or if already generated locate where it is stored, and direct the uploading of the generated hash table and/or index, etc. These type of instructions may appear as user options at the GUI that the user can select the chip to perform.

[0639] Furthermore, a library may be included wherein the library may include pre-existing, editable, configuration files, such as files orientated to the typical user selected functioning of the hardware, such as with respect to a portion or whole genome analysis, for instance, for ancestry analysis, or disease diagnostics, or drug discovery, or protein profiling, etc. These types of preset parameters, such as for performing such analyses, may be stored in the library. For example, if the platform herein described is employed such as for oncology research, the preset parameters may be configured differently than if the platform were directed simply to researching a genealogy.

[0640] More particularly, for oncology, accuracy may be an important factor, therefore, the parameters of the system may be set to ensure increased accuracy albeit in exchange for possibly a decrease in speed. However, for other genomics applications, speed may be the key determinant and therefore the parameters of the system may be set to maximize speed, which however may sacrifice some accuracy. Accordingly, in various embodiments, often used parameter settings for performing different tasks can be preset into the library to facilitate ease of use. Such parameter settings may also include the necessary software applications employed in running the system. For instance, the library may contain the code that executes the API, and may further include sample files, scripts, and any other ancillary information necessary for running the system. Hence, the library may be configured for compiling software for running the API as well as various executables.

[0641] In various instances, the PCI **2** (**104**) and/or chip **7** (**100**) may also include a memory, such as a Random Access Memory (RAM) or a Dynamic Rapid Access Memory with e.g. a DDR3 interface, such as a memory that may be used for facilitating the performance of the various modules described herein, for instance, the mapper, aligner, and/or sorter. For example, the DRAM may be where the reference, the hash table, and/or the hash table index, and/or reads may be stored. Further, the memory may be used for facilitating the performance of various other modules described herein, for instance, the deduper, local realigner, base quality score recalibrator, variant caller, compressor, and/or decompresor. For example, the DRAM may be where sorted reads, annotated reads, compressed reads, and/or variant calls may be stored. Further, the memory may be configured so as to include a separate interface for each of the various memory modules employed by the aligner and/or any other module, such as where each memory may include a file layer and logical layer. As indicated above, because there may be multiple memories and/or multiple modules, a chip level interconnect may be included so as to facilitate communication through the chip **7** (**100**).

[0642] Accordingly, in various instances, an apparatus of the disclosure may include a chip, wherein the chip includes an integrated circuit that is formed of a set of hardwired digital logic circuits that may be interconnected by one or more physical electrical interconnects. In various embodiments, the one or more physical electrical interconnects include an input to the integrated circuit that may be connected with an electronic data source for receiving data. Further, in certain embodiments, the hardwired digital logic circuits may be arranged as a set of processing engines, such as wherein each processing engine may be formed of a subset of the hardwired digital logic circuits, which are configured to perform one or more of the steps in the sequence analysis pipeline. More particularly, each subset of the hardwired digital logic circuits may be in a wired configuration so as to perform the one or more steps in the sequence analysis pipeline.

[0643] In various instances, the set of processing engines may include one or more of a mapping module, an alignment module, and/or a sorting module, such as where the one or more of these modules are in the wired configuration. For instance, a mapping module may be included, where in the wired configuration, the mapping module may access an index, such as of one or more genetic reference sequences, e.g., from a memory, such as via one or more of the plurality of physical electronic interconnects, so as to map the plurality of reads to one or more segments of the one or more genetic reference sequences. Further, in various instances, an alignment module may be included, wherein the wired configuration, the alignment module may access the one or more genetic reference sequences, e.g., from the memory, such as via one or more of the plurality of physical electronic interconnects, so to align the plurality of reads to the one or more segments of the one or more genetic reference sequences. Further still, in various instances, a sorting module may be included, wherein the wired configuration, the sorting module may access the one or more aligned sequences, e.g., from the memory, such as via one or more of the plurality of physical electronic interconnects, so to sort the plurality of reads to a chromosome, such as from the one or more genetic reference sequences. In like manner, in various instances, one or more of local realignment, duplicate marking, base quality score recalibration, and/or variant calling modules may be included in the chip, such as in the wired configuration consistent as with the modules described above, so as to perform their respective functions.

[0644] As indicated above, in various instances one or more integrated circuits of the disclosure may be configured as one or more chips such as one or more of an ASIC, a FPGA, and/or a structured ASIC chip. For instance, an integrated circuit is characteristically a set of electronic circuits on a small wafer or "chip" of semiconductor material, such as silicon. Typically integrated circuits include circuit elements that may be inseparably associated and electrically interconnected. A prototypical digital integrated circuit includes a variety of circuit elements such as one or more of logic gates, flip-flops, multiplexers, and other various circuit elements that are configured and/or configurable for functioning in circuit such as a microprocessor, or other microcontroller, such as for binary processing of "zero" and "one" signals, for instance, in the performance of one or more of the operations of the disclosure.

[0645] More particularly, one or more mask-programmable logic gates may be configured or programmed for performing a logical operation, such as implementing a Boolean function, on one or more logical inputs so as to produce a single logical output. Such logic gates may be configured using one or more diodes or transistors in such a manner that the gate operates as an electronic switch. In various instances, logic gates can be cascaded in a manner akin to the way that Boolean functions can be composed, thereby allowing the construction of a physical model of all of Boolean logic and, therefore, all of the algorithms and mathematics that can be described with Boolean logic, such as those described herein, may be implemented in the logic gates of the integrated circuits of the present disclosure. In various embodiments, a collection of gates may be present on the wafer in such a manner as to form a gate array, such as a gate array circuit.

[0646] In various instances, an integrated circuit may also include one or more flip-flops. A flip-flop may be a circuit, or at least a part thereof, that is configured as a latch. Typically, a flip-flop has two stable states and can change from one to the other such as by signals applied to one or more control inputs, and, therefore, a flip-flop will have one or two outputs. In use, flip-flops are employed to store state information, and consequently, may be deployed as a basic storage element, such as in sequential logic operations. The integrated may also include a multiplexer. A multiplexer may be configured for selecting one of several input signals, such as digital (or analog) input signals, and further may be configured for forwarding the selected input to an output. In this manner, a multiplexer may be used to increase the amount of data that can be sent over a network within a certain amount of time and bandwidth.

[0647] In certain instances, as recited herein, a typical integrated circuit can include anywhere from one to millions of such circuit elements configured for performing operations, such as those operations presently disclosed, wherein the various circuit elements occupy only a few square millimeters of space. The small size of these circuits allows high speed, low power dissipation, and reduced manufacturing cost.

[0648] Such integrated circuits may be fabricated using a variety of different technologies but, in general, are usually constructed as a monolithic integrated circuit. For instance, a typical integrated circuit, e.g., a semiconductor, may be fabricated in a layer process, such as a layer process that includes about three main process steps, such as imaging, deposition and etching. In various instances, one or more of these process steps may be supplemented by further processing steps such as doping, cleaning, and the like. For example, in a typical fabrication procedure, a wafer, such as a mono-crystal silicon wafer may be provided for use as a substrate upon which the integrated circuit is to be constructed, e.g., printed. Photolithography may then be employed to print on the wafer so as to mark different areas of the substrate that may then be doped and/or printed with tracks, such as with a metal insulator such as aluminum.

[0649] Typically, an integrated circuit is composed of one or a plurality of overlapping layers, such as where each layer is defined by photolithography. Some layers may form diffusion layers, marking where various dopants have diffused into the substrate, and other layers define where additional ions may be implanted. Additional layers may define the conductors (e.g., polysilicon, metal layers, and the like) as well as the connection layers between the conducting layers. For instance, a transistor may be formed wherever the gate layer (polysilicon or metal) crosses a diffusion layer, and in various instances, meandering stripes may be used to form on-chip resistors. Exemplary integrated circuits may include: an ASIC, an FGPA, and/or a Structured ASIC.

[0650] Often times, integrated circuits are fabricated for general use. However, in various instances, such as some of those described herein, an integrated circuit may be customized, such as to form an application-specific integrated circuit or "ASIC." An ASIC, generally referred to as a "standard cell ASIC," is an integrated circuit that has been customized for a particular use, rather than for a general-purpose use. Typically an ASIC may have a large number of logic gates, such as in some instances, over 100 million gates, which gates can be configured for preforming a multiplicity of different operations such as being configured as microprocessors and/or memory blocks, including ROM, RAM, EEPROM, flash memory, and other large building blocks, such as for the purpose of performing the operations herein disclosed. A unique feature of an ASIC is that because it is a chip that is constructed for performing a specific set of applications, the chip may be fabricated in such a manner as to be customizable, such as by employing a gate-array design protocol.

[0651] For instance, a gate array or uncommitted logic array (ULA) may be used in the design and manufacture of application-specific integrated circuits (ASICs). In such an instance, an ASIC may be manufactured from a prefabricated chip that has active devices like gates, e.g., NAND-gates, which at first may be unconnected, but may at a later time be interconnected, such as according to the gate-array design protocol, for example, by adding metal layers, such as in the factory. Accordingly, with respect to producing an ASIC, a gate array circuit may be prefabricated on a silicon chip circuit that upon production has no particular function, but does include one or more of transistors, standard NAND or NOR logic gates, and may have further other active devices that may be placed at predefined positions and manufactured on the wafer, which wafer in this instance may be termed a "master slice." Hence, the creation of a circuit having the determined specified functions may be accomplished by adding a final surface layer or layers of metal interconnects to the chips on the master slice late in the manufacturing process, and joining these elements to allow the function of the chip to be customized as desired, e.g., in accordance with the design protocol.

[0652] More particularly, a gate-array design protocol employs a manufacturing method where the various diffused layers, e.g., transistors and other active circuit elements, such as those described above, are predefined and constructed on general use wafers but are stored prior to metallization such that various of the circuit elements remain unconnected. In such an instance, the chip may then, at a later point in time, be customized in accordance with various specific use parameters such as by a physical design process that defines the interconnections of the final device. For instance, gate array master slices are usually prefabricated and stockpiled in large quantities waiting for customization. An application circuit must be built on the gate array in such a manner that the circuit has enough gates, wiring and I/O pins so as to perform the desired functions.

[0653] Since requirements vary, gate array wafers often come in standard families, including larger members having more, e.g., all, resources, but being correspondingly more expensive, and somewhat smaller members having a limited selection of resources, but also being less expensive. The right wafer standard should be chosen based on the number of resources required to perform the selected functions. The amount of resources to be deployed may fairly easily be determined, such as by counting how many gates and I/Os pins are needed, however, the amount of routing tracks needed may vary considerably and should therefore be selected carefully. However, because the master slice is somewhat prefabricated, the design and fabrication, according to the individual design protocol specifications, may be finished in a shorter time compared with standard cell or full custom (FPGA) design. In a manner such as this, the gate array approach reduces the mask costs, since fewer custom masks need to be produced. In addition manufacturing test tooling lead time and costs are also reduced, since the same test fixtures may be used for all gate array products manufactured on the same die size.

[0654] In such an instance, the manufacture of such a standard cell ASIC may include anywhere from two to nine, or ten, or twelve, or more deposition layers, such as where one or more, e.g., all, of the subsequent metal layers run perpendicular to the one below it. Such fabrication methods are useful because they provide for a somewhat customized chip design in a relatively short construction time period because the final metallization process can be performed quickly. However, such gate-array ASICs are often a compromise as mapping a given design onto a "stock" wafer does not typically give 100% utilization. Another disadvantage with respect to an ASIC is the non-recurring engineering (NRE) cost that can run into the millions of dollars. Nevertheless, the per unit production cost of an ASIC can be quite low, comparatively.

[0655] An alternative to a standard cell ASIC for the production of customizable chips is a field-programmable gated array or "FPGA." An FPGA employs programmable logic blocks and interconnects that are re-writeable thereby allowing the same FPGA to be designed and at least partially re-designed so as to be used in many different applications, or the same applications in a multiplicity of different ways over time. More specifically, a field-programmable gate array is an integrated circuit that is designed to be configured one or a multiplicity of times, such as by a customer or a designer, e.g., after manufacturing.

[0656] Typically, FPGAs have large resources of logic gates and/or memory, e.g., RAM, blocks that can be configured to implement complex digital computations. For instance, FPGAs contain programmable logic components called "logic blocks", as well as a multiplicity, e.g., a hierarchy, of reconfigurable interconnects that allow the blocks to be "wired together." More particularly, FGPAs may have a multiplicity of changeable logic gates that can be inter-wired in a variety of different configurations, so as to form logic blocks that can be configured to perform a wide variety of complex combinational functions, such as those with respect to performing the operations herein detailed. In various instances, the logic blocks of an FPGA may be configured to include memory elements such as simple flip-flops or more complete memory blocks such as ROM or RAM. As FPGA designs employ very fast I/Os and bidirectional data buses it may, in certain instances, be difficult to verify the correct timing of valid data within setup and hold times. Accordingly, in some instances, the appropriate floor planning may enable resource allocations within an FPGA to meet these time constraints. FPGAs, therefore, may be used to implement any logical function that a standard cell ASIC could perform. However, the ability to update the functionality after shipping, partial re-configuration of a portion of the design, and the low non-recurring engineering costs relative to an ASIC design (notwithstanding the generally higher per unit cost), offer advantages for many applications.

[0657] In some instances, the coarse-grained architectural approach of a typical FPGA fabrication may be performed in such a manner as to combine the logic blocks and interconnects of traditional FPGAs with embedded microprocessors and related peripherals to form a complete "system on a programmable chip". In certain instances, an FPGA of the disclosure may have the ability to be reprogrammed at "run time," and may, in accordance with the methods disclosed herein, allow for reconfigurable computing or the production of reconfigurable systems, e.g., a CPU that can reconfigure itself to suit the operations disclosed herein. In some instances, software-configurable microprocessors may be employed to provide an array of processor cores and FPGA-like programmable cores that may be present on the same chip.

[0658] A common FPGA architecture may include an array of configurable logic blocks, I/O pads, and/or one or more routing channels. Typically, a logic block may include one or a plurality of logical cells, where a typical cell may include a 4-input LUT, a Full adder (FA), and/or flip-flop, and the like, which function to produce an output. In various instances, the output can be either synchronous or asynchronous. An application circuit may be mapped into an FPGA and the number of logic blocks, I/Os, and routing tracks to be included can be determined from the design, the number of which may vary. It is to be noted that since unused routing tracks may increase the cost and decrease the performance of the integrated circuit without providing any benefit, the number of routing tracks should be enough such that its processes fit in terms of lookup tables (LUTs) and I/Os to be routed without being in excess. Further, since clock signals are normally routed via special-purpose dedicated routing networks (e.g., global buffers) they and other such signals may be separately managed.

[0659] An FPGA, as herein disclosed, may also include higher level functionality fixed into the silicon, such as one or more multipliers, generic DSP blocks, embedded processors, high speed I/O logic, and/or embedded memories. Inclusion of these common functions embedded into the silicon wafer reduces the area required and gives those functions increased speed. It is to be noted that the disclosed FPGAs may be used for systems validation including pre-silicon validation, post-silicon validation, and firmware development, such as to validate the final design prior to the production of "for use" chips, such as standard cell ASIC or Structured ASIC chips, which may represent the final end product.

[0660] In the production of an exemplary integrated circuit, such as an FPGA, etc., having the requisite functionality as herein described, one or more of the following steps may be followed, in any logical sequence. First, a hardware description language (HDL) or a schematic design may be provided. An electronic design automation tool, e.g., a CAD, can then be employed to generate a technology-mapped netlist. The netlist can then be fitted to the actual FPGA architecture such as by using a process called place-and-route in accordance with the appropriate place-and-route software. Once the design and validation process is complete, the binary file generated may be used to (re)configure the FPGA.

[0661] In a typical design protocol flow, the design may be simulated at multiple stages throughout the design process. Initially the RTL description, such as in VHDL or Verilog, may be simulated by creating test benches to simulate the system and observe results. In certain instances, the synthesis engine may map the proposed design to the netlist, and after the synthesis engine has mapped the design to a netlist, the netlist may be translated to a gate level description. At this stage a simulation may be performed, e.g., again, to confirm the synthesis proceeded without errors. The design may then be laid out in the FPGA, at which point propagation delays may be added, and a simulation may be run, e.g., again, with these values back-annotated onto the netlist, such as prior to final validation and further fabrication, such as in the generation of one or more ASIC or structured ASIC based chips.

[0662] Accordingly, a hybrid between an ASIC and a FPGA is a structured ASIC, which falls between an FPGA and an ASIC. The traditional "standard cell ASIC", disclosed above, is typically expensive, e.g., extremely expensive, and time consuming to develop. For instance, in developing a standard cell ASIC a large set of photolithographic masks may be produced for each standard cell ASIC design. However, after this up-front investment in the initial development has been made, the typical production costs become very low, and the operating parameters with respect to power, frequency, and logic capacity can readily be optimized.

[0663] Alternatively, unlike Standard cell ASICs, the typical FPGA and/or CLPD, containing programmable logic, are relatively fast and cheap to develop, largely because the pre-existing devices are programmed electronically, and no photolithographic masks are required. However, with respect to operating parameters, such as power, frequency, and logic capacity, these are poor in comparison to a standard cell ASIC, and per-unit costs can be very high, particularly for large-capacity devices.

[0664] Structured ASICs, on the other hand, are a compromise between these two. Unlike gate arrays, structured ASICs tend to include predefined or configurable memories and/or analog blocks. Hence, development cost is much lower than for standard cell, because only a few photolithographic masks must be produced for each structured ASIC design, such as for configurable metal layers. And, although per-unit production costs are significantly higher than standard cell, they are still far lower than FPGA unit costs. With respect to power and frequency, these are a compromise between standard cell and FPGAs, but their logic capacity is similar to the largest FPGAs. Hence, in many instances, structured ASICs may be a technology that can reduce the up-front cost and time to develop a new custom integrated circuit.

[0665] With respect to design and fabrication of a structured ASIC, and as shown in FIG. **35**, before a series of structured ASICs can be developed, a "master slice" **902** may first be developed, such as by using standard cell ASIC methodology. As indicated above, the master slice may include most of the typical integrated circuit layers, such as one or more transistors, memories or memory cells, input/output cells, phase-locked loops, or other clock generators, and the like. Optionally a master slice may contain flip-flops, latches, and/or multi-transistor combinational gates. Some amount of local wiring between components may be included in the master slice, but much of the wiring to implement a full logic design may be omitted, such as to be added later. Note that a master slice can theoretically be constructed to include any logic suitable for standard cell ASICs, potentially including large complex modules, and operating parameters (power, frequency, logic capacity) of master slice logic are optimal, just as for standard cell ASICs. Photolithographic masks may be produced for master slice content, the mask set being similar or somewhat smaller than a standard cell ASIC mask set. Accordingly, the master slice **902** includes a set of digital logic circuits **903** that may or may not yet be hardwired to function in a particular way.

[0666] Following construction of the master slice **903**, a series of one or more complete structured ASICs may be implemented, such as by building upon the same master slice. Typically many structured ASIC designs utilize the same master slice, to amortize the cost of the master slice over many projects. Each individual structured ASIC design may be implemented by determining a set of new wired connections between components (transistors, etc.) in the master slice, which will effectively build the master slice components into higher level gates, flip flops, latches, memories, and large complex logic modules. Accordingly, these determined wired connections **905** may be implemented in a small number of additional "configurable" metal layers **904**A and **904**B fabricated on top of the master slice, such as by connecting metal pads, or vias, in the master slice, for instance, by wires in the configurable metal layers. These additional metal layers are called "configurable" because they can be customized to each structured ASIC design project; however, they are fixed at fabrication time and cannot be rewired electronically except as the implemented logic design provides. There can be any number of configurable metal layers **904**.

[0667] Most any conceivable logic design can thus be implemented using a master slice and appropriate wiring metal layers, as long as the master slice contains enough logic resources (transistors, memories, etc.) to form all the required logic design elements. The number of configurable metal layers varies from one structured ASIC design flow to another, but typically may be between 1 and 5 configurable metal layers more or less. A small additional set of photolithographic masks may be produced, corresponding to the configurable metal layers, and in device fabrication, the full mask set (master slice masks and configurable metal layer masks) may be used to build wafers of complete structured ASIC dice. Alternatively, master slice wafers might be pre-fabricated in bulk, and metal layers added in a later fabrication step to complete wafers of specific structured ASIC designs.

[0668] Advantageously, a structured ASIC master slice can be designed in one step, e.g., by a first designer, while specific structured ASIC logic designs based on that master slice may be designed, in a second step, such as by various other designers utilizing services of the structured ASIC designer. In particular, the various parties may typically be responsible for "front end" logic design specific to the desired integrated circuit functionality, such as RTL (register transfer logic) code development, simulation, emulation, regression testing, debugging, and the like; while the structured ASIC designer may typically be responsible for "back end" design flow, including synthesis, place and route, static timing analysis, test logic insertion, and/or tapeout. An additional party, e.g., a foundry, may be employed to produce physical photolithographic masks, fabricate wafers, and/or test and/or package the device dice. In various instances, a structured ASIC designer may also design custom master slices for a particular application class, such as to contain logic resource types or quantities customized to those applications.

[0669] Accordingly, by virtue of there being pre-defined metal layers (thus reducing manufacturing time) and pre-characterization of what is on the silicon wafer, e.g., master slice, (thus reducing design cycle time) the cycle time and design cycle time in the structured ASIC may be reduced as compared to typical ASIC manufacturing processes. For instance, in a cell based ASIC design or FPGA, e.g., gate-array, design the user may often have to design power, clock, and test structures themselves. However, in a structured ASIC these may be predefined which can save production time and expense as compared to cell based or gate-array profiles.

[0670] Particularly, the design task for structured ASIC's is to map the circuit into a fixed arrangement of known cells. More particularly, the comparative architecture of a structured ASIC typically may include two main levels, such as both structured elements and an array of structured elements.

Such structured elements may include both combinational and sequential function blocks, which can function as either logical or storage elements. Additionally, with respect to arrays of structural elements, uniform or non-uniform array styles may be employed such as in a fixed arrangement of structured elements.

[0671] Consequently, in a structured ASIC design, the logic mask-layers of the device may be predefined. In such an instance, design differentiation and customization may be achieved such as by creating custom metal layers that create custom connections between predefined lower-layer logic elements. Likewise, the design tools used for structured ASIC can be substantially lower in cost and easier (faster) to use than cell-based tools, because they do not have to perform all the functions that cell-based tools do. More particularly, pre-existing standard cell-based CAD tools may be used in the design process. In some instances, however, CAD tools designed specifically for structured ASIC's may be used. Product specific placement tools may also be used. Further, as disclosed herein, new and improved algorithms have been developed so as to exploit the modularity of structured ASIC's, and better account for a more clock aware design. Additionally, the methods herein disclosed may be employed so as to enhance the evaluation and analysis processes, as discussed above

[0672] In these manners the structured ASIC technology may act as a bridge filling the gap between field-programmable gate arrays and standard ASIC designs. More specifically, because only a small number of chip layers need be custom-produced, structured ASIC designs may have much smaller non-recurring expenditures (NRE) than "standard-cell" or "full-custom" chips, which require that a full mask set be produced for every design. Accordingly, a structured ASIC offers high performance (a characteristic of a typical ASIC), and low NRE cost (a characteristic of FPGA). Hence, a Structured ASIC fabrication process can be employed so as to allow the end product to be introduced quickly to market, to have lower cost, and to be more easily designed.

[0673] In some instances, however, a FPGA, may be advantageous in that the interconnects and logic blocks are programmable after fabrication. This offers a high flexibility of design and ease of debugging in prototyping. However, the capability of FPGAs to implement large circuits is sometimes limited, in both size and speed, which in some circumstances, may be due to the inherent complexity in programmable routing and/or significant space that may be occupied by the various included programming elements. On the other hand, ASICs also have some disadvantages, such as an expensive design flow, due in part to the fact that every different design typically needs a complete different set of masks. The structured ASIC, therefore, may be a solution between these two. It may basically have the same structure as a FPGA, but may be mask-programmable, such as in an ASIC, instead of being field-programmable, by configuring one or several via layers between metal layers. For instance, one or more, e.g., each SRAM configuration bit can be replaced by a choice of either including or not including a via or between various metal contacts.

[0674] For example, with respect to the architecture of a structured ASIC, a typical architecture may often times be fine-grained, medium grained, and/or hierarchical. A fine-grained architecture may include many connections in and out of a structured element, whereas higher granularities reduce connections to the structured element but may also decrease the functionality it can support. Each individual design will benefit differently at varying granularities. More particularly, in a fine-grained architecture, the architecture may include structured elements that contain unconnected discrete components, such as transistors, resistors, and other control elements that can later be connected. In a medium grained architecture, the architecture of the structured elements may include generic logic as well as gates, MUX's, LUT's and/or storage elements, such as flip-flops. Alternatively, in a hierarchical architecture, the architecture may include mini structured elements, for instance that contain gates, MUX's, and LUT's, but do not typically contain storage elements like flip-flops. In other instances, the mini element may be combined with registers or flip-flops.

[0675] With respect to implementing a structured ASIC the various fabrication steps may include one or more of register transfer level design (RTL); logical synthesis, so as to map the RTL into structured elements; design for test insertion, so as to improve testability and fault coverage; placement, so as to map each structured element onto an array element and to place each element into a fixed arrangement; physical synthesis in such a manner that improves the timing of the layout, and optimizes the placement of each element; clock synthesis in a manner that distributes the clock network and minimizes the clock skew and delay; as well as routing or otherwise inserting the wiring between the various elements. In various instances, these steps may be performed in any logical order and in a manner to make the design process, such as with respect to logical synthesis, less complex, as well as to help build up a more complete target structured ASIC library that enhances what specifically can be implemented from the design.

[0676] Furthermore, it has become common for some designers of processor cores to license the processor design to various customers so as to embed in their own silicon devices. Such embedded cores may include ARM, PowerPC, Krait, etc. as general-purpose processors, and may also include more specialized processors such as graphics processors (GPUs) or vector processors. Embedded processor cores may be large, complex logic modules, pipelined to run at high operating frequencies such as about 1 or 2 GHz to about 3 to 6 GHz, or more. In order to achieve such high frequencies, careful physical layout and routing may be used for processor cores and associated cache memory; and as a result, embedded processor technology may often be supplied as a "hard macro" (such as for defining precise placement and routing of the subcomponents) for a particular silicon fabrication process.

[0677] However, such an embedded processor core may be a suboptimal candidate for implementation in a structured ASIC using configurable metal layers. Hard macros do not generally apply to structured ASIC configurable layers, and even if an embedded processor were implemented as closely as possible to its hard macro in the configurable metal layers, it would likely be frequency limited (e.g. 30% or 50% of nominal operating frequency), and would likely consume very large portions of the available master slice resources. The relative area inefficiency of structured ASIC fabric as compared to standard cell could cause the embedded processor to cover a significantly larger physical silicon area, and in combination with reduced operating frequency, the performance to area (or cost) ratio could be much lower than a standard cell implementation of the same embedded core.

[0678] However, it is practical to implement embedded one or more processor cores efficiently in a structured ASIC master slice, such as by using a standard cell design methodology, as disclosed herein, including the use of hard macros. These would retain full operating frequency and performance, and consume only normal silicon area. The processor core and/or cache input and output wires could be connected to other resources in the master slice, or advantageously, exposed to configurable metal layer routing, to enable the embedded cores to be connected to any infrastructure and logic modules implemented in each particular structured ASIC design. In a manner such as this, the embedded processor cores become master slice resources available to many various structured ASIC designs later implemented using the master slice.

[0679] Embedded processor cores in a structured ASIC can be connected to logic infrastructure so that software (firmware) running on the cores can share and access various memory and other resources, on-chip and off-chip, and to communicate with any or all other logic modules on the chip, via memory and/or directly. In this manner, the processor cores can operate in parallel with other logic modules, and/or cooperate with other logic modules to complete joint work, such as by the processor cores requesting tasks to be performed by other modules, or other modules requesting tasks to be performed by the processor cores, or both.

[0680] When Bio-IT acceleration modules (such as to perform mapping, alignment, sorting, duplicate marking, base quality score recalibration, local re-alignment, variant calling, compression, decompression, etc. as described herein) are implemented in a structured ASIC along with embedded processor cores, the resulting system on a chip (SOC) has important advantages, especially in a combination of speed and flexibility. Extreme speed may be achieved by the hardware acceleration modules, and extreme flexibility may be achieved by the full programmability of the processor cores. By reprogramming the processor cores, the bio-IT algorithms executed can be easily modified, but these algorithms can run orders of magnitude faster than in traditional CPUs because computationally intensive operations may be offloaded to hardware accelerators. Communication and memory organization can be optimized for cooperative processor-accelerator work. Additional software algorithm acceleration can be obtained by additional hardware modules designed to pre-process or post-process data used by the processor cores, such as organizing reads overlapping a reference genome locus into a pileup data structure, for presentation to the processor cores. In some processor architectures, instruction sets can be extended to utilize connected hardware resources; in the Bio-IT SOC environment, new processor instructions can be defined to access Bio-IT hardware acceleration functions.

[0681] As summarized in TABLE II, below, a structured ASIC, therefore, has several prefabricated advantages, such as over an ASIC or FPGA. For instance, the various components may be "almost" connected, such as in a variety of predefined configurations, and multiple global and local clocks may be prefabricated. This means, therefore, that signal integrity and timing issues should inherently be addressed. Additionally, only a few metal layers may be needed for fabrication. Further, unlike standard FPGAs, the structured ASIC should have a capacity, performance, and power consumption closer to that of a standard cell ASIC. This should allow for easier and faster design processes and times as well as reduced NRE costs than in standard cell ASIC's, and should drastically reduce turnaround time. Further still, no skew problems should need to be addressed.

TABLE II

	FPGA	Structured ASIC	Standard Cell ASIC
Silicon area	Very high	Low	Very low
Power utilization	High	Low	Very low
Operating frequency	Low	High	High
Logic capacity	Medium	Medium	High
Development cost	Very low	Low	High
Per-unit cost	Very high	Low	Very low

[0682] A structured ASIC, therefore, has several different beneficial properties, including one or more of: low NRE cost, lower requirements for implementation engineering efforts, lower mask tooling charges, such as over an ASIC, with the additional benefits of high performance, low power consumption, fewer fabrication layers, less complexity, in a pre-made cell block configuration that is available for placing circuit elements, together which leads to a quicker production time. There are, however, some disadvantages to structured ASICs, for instance, there are sometimes a lack of adequate design tools, which tools and processing may be expensive and need to be altered from traditional ASIC tools. Further, these new architectures are still being subjected to formal evaluations and comparative analyses. And, there may be tradeoffs between 3-, 4-, and 5-input LUT's, and/or between sizes of distributed RAM.

[0683] Accordingly, in view of the above, there are both advantages and disadvantages to ASICs, FPGAs, and Structured ASICs. For instance, standard cell ASICs may be difficult to design, need a long development time, have a high NRE cost. However, an ASIC may also support large designs, support complex designs, have a high performance at a low power consumption, which therefore could result in a low or lower Per-Unit Cost (at high volume). On the other hand, FPGAs may be easy to design, involve a short development time, and a low NRE cost. However, FPGAs may have a limited design size and/or complexity, may have limited performance, and a high power consumption, which may result in a high or higher Per-Unit Cost. In many instances, a structured ASIC may be designed to maximize these benefits and minimize these disadvantages. For instance, generally speaking there may be about a 100:33:1 ratio between the number of gates in a given area for standard cell ASIC's, structured ASIC's, and FPGA's; a 100:75:15 ratio for performance (based on clock frequency); and a 1:3:12 ratio for power, respectively.

[0684] As indicated above, in various instances a chip of the disclosure may be configured as an expansion card, such as where the chip includes a PCIe bus and is positioned so as to be in communication with one or more memories, such as being surrounding by memories, such as being substantially surrounded by memories, such as being entirely surrounded by memories. In various embodiments, the chip may be a dense and/or fast FPGA chip that in various instances, may be convertible to an ASIC or an sASIC. In various instances, the chip may be a structured ASIC that is convertible into an ASIC. In some instances, the chip may be an ASIC.

[0685] As indicated above, the modules herein disclosed may be implemented in the hardware of the chip, such as by being hardwired therein, and in such instances their implementation may be such that their functioning may take place at a faster speed as compared to when implemented in software, such as where there are minimal instructions to be fetched, read, and/or executed. Hence, given the unique hardware implementation, the modules of the disclosure may function directly in accordance with their operations parameters, such as without needing to fetch, read, and/or execute instructions. Additionally, memory requirements and processing times may be reduced, such as where the communications within chip is via files rather than through accessing a memory. Of course, in some instances, the chip and/or card may be sized so as to include more memory, such as more on board memory, so as to enhance parallel processing capabilities, thereby resulting in even faster processing speeds. For instance, in certain embodiments, a chip of the disclosure may include an embedded DRAM, so that the chip does not have to rely on external memory, which would therefore result in a further increase in processing speed, such as where a Burrows-Wheeler algorithm may be employed, instead of a hash table and hash function, which may in various instances, rely on external, e.g., host memory. In such instances, the running of the entire pipeline can be accomplished in 6 minutes or less, such as from start to finish.

[0686] As indicated above, there are various different points where any given module can be positioned on the hardware, or be positioned remotely therefrom, such as on a server accessible on the cloud. Where a given module is positioned on the chip, e.g., hardwired into the chip, its function may be performed by the hardware, however, where desired, the module may be positioned remotely from the chip, at which point the platform may include the necessary instrumentality for sending the relevant data to a remote location, such as a server accessible via the cloud, so that the particular module's functionality may be engaged for further processing of the data, in accordance with the user selected desired protocols. Accordingly, part of the platform may include a web-based interface for the performance of one or more tasks pursuant to the functioning of one or more of the modules disclosed herein. For instance, where mapping, alignment, and/or sorting are all modules that may occur on the chip, in various instances, one or more of local realignment, duplicate marking, base quality core recalibration, and/or variant calling may take place on the cloud.

[0687] Particularly, once the genetic data has been generated and/or processed, e.g., in one or more primary and/or secondary processing protocols, such as by being mapped, aligned, and/or sorted, such as to produce one or more variant call files, for instance, to determine how the genetic sequence data from a subject differs from one or more reference sequences, a further aspect of the disclosure may be directed to performing one or more other analytical functions on the generated and/or processed genetic data such as for further, e.g., tertiary, processing, as depicted in FIG. **33**. For example, the system may be configured for further processing of the generated and/or secondarily processed data, such as by running it through one or more tertiary processing pipelines **700**, such as one or more of a genome pipeline, an epigenome pipeline, metagenome pipeline, joint genotyping, a MuTect2 pipeline, or other tertiary processing pipeline, such as by the devices and methods disclosed herein. For instance, in various instances, an additional layer of processing **800** may be provided, such as for disease diagnostics, therapeutic treatment, and/or prophylactic prevention, such as including NIPT, NICU, Cancer, LDT, AgBio, and other such disease diagnostics, prophylaxis, and/or treatments employing the data generated by one or more of the present primary and/or secondary and/or tertiary pipelines. Hence, the devices and methods herein disclosed may be used to generate genetic sequence data, which data may then be used to generate one or more variant call files and/or other associated data that may further be subject to the execution of other tertiary processing pipelines in accordance with the devices and methods disclosed herein, such as for particular and/or general disease diagnostics as well as for prophylactic and/or therapeutic treatment and/or developmental modalities.

[0688] As described above, the system herein presented may include the generating, such as by the sequencer on a chip technology as disclosed herein, or the otherwise acquiring of genetic sequence data, and may include the performance of one or more secondary processing protocols, such as including one or more of mapping, aligning, and sorting of the generated genetic sequence data, such as to produce one or more variant call files, for instance, so as to determine how the genetic sequence data from a subject differs from one or more reference sequences or genomes. A further aspect of the disclosure may be directed to performing one or more other analytical functions on the generated and/or processed genetic data such as for further, e.g., tertiary, processing, which processing may be performed on or in association with the same chip or chipset as that hosting the aforementioned sequencer technology.

[0689] In a first instance, such as with respect to the generation, acquisition, and/or transmission of genetic sequence data, as set forth in FIG. **30**, such data may be produced either locally or remotely and/or the results thereof may then be directly processed, such as by a local computing resource **100**, or may be transmitted to a remote location, such as to a remote computing resource **300**, for further processing. For instance, the generated genetic sequence data may be processed locally, and directly, such as where the sequencing and secondary processing functionalities are housed on the same chipset and/or within the same device. Likewise, the generated genetic sequence data may be processed locally, and indirectly, such as where the sequencing and secondary processing functionalities occur separately by distinct apparatuses that share the same facility or location but may be separated by a space albeit communicably connected, such as via a local network **10**. In a further instance, the genetic sequence data may be derived remotely, such as by a NGS, and the resultant data may be transmitted over a cloud based network **50** to a remote location, such as separated geographically from the sequencer.

[0690] Specifically, as illustrated in FIG. **30**, in various embodiments, a nucleotide sequencer may be provided on site, such as by a sequencer on a chip or by an NGS, wherein the sequencer is associated with a local computing resource **100** either directly or indirectly such as by a local network connection **10**. The local computing resource **100** may include or otherwise be associated with one or more of a data generation **110** and/or a data acquisition **120** mechanism(s). Such mechanisms may be any mechanism configured for either generating and/or otherwise acquiring data, such as analog, digital, and/or electromagnetic data related to one or more genetic sequences of a subject or group of subjects.

[0691] For example, such a data generating mechanism **110** may be a primary processor such as a sequencer, such as a NGS, a sequencer on a chip, or other like mechanism for generating genetic sequence information. Further, such data acquisition mechanisms **120** may be any mechanism configured for receiving data, such as generated genetic sequence information, and/or together with the data generator **110** and/or computing resource **150** capable of subjecting the same to one or more secondary processing protocols, such as a secondary processing pipeline apparatus configured for running a mapper, aligner, sorter, and/or variant caller protocol on the generated and/or acquired sequence data as herein described. In various instances, the data generating **110** and/or data acquisition **120** apparatuses may be networked together such as over a local network **10**, such as for local storage **200**, or may be networked together over a cloud based network **30**, such as for transmitting and/or receiving data, such as digital data related to the primary and/or secondary processing of genetic sequence information, such as to or from a remote location **30** such as for remote processing **300** and/or storage **400**. In various embodiments, one or more of these components may be communicably coupled together by a hybrid network as herein described.

[0692] The local computing resource **100** may also include or otherwise be associated with a compiler **130** and/or a processor **150**, such as a compiler **130** configured for compiling the generated and/or acquired data and/or data associated therewith, and a processor **150** configured for processing the generated and/or acquired and/or compiled data and/or controlling the system **1** and its components as herein described. Further, the local computing resource **100** may include a compressor unit **160** configured for compressing data, such as generated and/or acquired primary and/or secondary processed data, which data may be compressed, such as prior to transfer over a local **10** and/or cloud **30** and/or hybrid cloud based **50** network.

[0693] In particular instances, the system **1** may be configured for subjecting the generated and/or secondarily processed data to further processing, e.g., via a local **100** and/or a remote **300** computing resource, such as by running it through one or more tertiary processing pipelines, such as one or more of a genome pipeline, an epigenome pipeline, metagenome pipeline, joint genotyping, a MuTect2 pipeline, or other tertiary processing pipeline. Such data may then be compressed and/or stored locally **200** and/or be transferred so as to be stored remotely.

[0694] In additional instances, the system **1** may include a further tier of processing modules, such as configured for rendering additional processing such as for diagnosis, disease and/or therapeutic discovery, and/or prophylaxis thereof. For instance, in various instances, an additional layer of processing may be provided, such as for disease diagnostics, therapeutic treatment, and/or prophylactic prevention, such as including NIPT, NICU, Cancer, LDT, AgBio, and other such disease diagnostics, prophylaxis, and/or treatments employing the data generated by one or more of the present primary and/or secondary and/or tertiary pipelines.

[0695] Accordingly, herein presented is a system **1** for producing and using a global hybrid cloud network **50**. For instance, presently, the cloud **30** is used primarily for storage, such as at a remote storage location **400**. In such an instance, the computing of data is performed locally **100** by a local computing resource **150**, and where storage needs are extensive, the cloud **30** is accessed so as to store the data generated by the local computing resource **150**, such as by use of a remote storage resource **400**. Hence, generated data is typically either wholly managed on site locally **100**, or it is totally managed off site **300**, on the cloud **30**.

[0696] Particularly, in a general implementation of a bioinformatics analysis platform, the computing **150** and/or storage **200** functions are maintained locally on site, and where storage needs exceed local storage capacity, or where there is a need for stored data to be made available to other remote users, such data may be transferred via internet **30** to the cloud for remote storage **400** thereby. In such an instance, where the computing resources **150** required for performance of the computing functions are minimal, but the storage requirements extensive, the computing function **150** may be maintained locally **100**, while the storage function **400** may be maintained remotely, with the fully processed data being transferred back and forth between the processing function **150**, such as for local processing only, and the storage function **400**, such as for the remote storage **400** of the processed data.

[0697] For instance, this may be exemplified with respect to the sequencing function, such as with a typical NGS, where the computing resource **150** is configured for performing the functions required for the sequencing of the genetic material so as to produce genetic sequenced data, e.g., reads, which data is produced onsite **100**. These reads, once generated, such as by the onsite NGS, may then be transferred such as over the cloud network **30**, such as for storage **400** at a remote location **300** in a manner so as to be recalled from the cloud **30** when necessary such as for further processing, such as for the performance of one or more of secondary and/or tertiary processing functions, that is at a location remote from the storage facility **400**, e.g., locally. In such an instance, the local storage resource **150** serves merely as a storage cache where data is placed while waiting transfer to or from the cloud **30**, such as to or from the remote storage facility **400**.

[0698] Likewise, where the computing function is extensive, such as requiring one or more remote computer cluster cores **300** for processing the data, and where the storage demands for storing the processed data **200** are relatively minimal, as compared to the computing resources **300** required to process the data, the data to be processed may be sent, such as over the cloud **30**, so as to be processed by a remote computing resource **300**, which resource may include one or more cores or clusters of computing resources, e.g., one or more super computing resources. In such an instance, once the data has been processed by the cloud based computer core **300**, the processed data may then be transferred over the cloud network **30** so as to be stored local **200** and readily available for use by the local computing resource **150**, such as for local analysis and/or diagnostics.

[0699] This may be exemplified with respect to a typical secondary processing function, such as where the pre-processed sequenced, e.g., read, data that is stored locally **200** is accessed, such as by the local computing resource **100**, and transmitted over the cloud internet **30** to a remote computing facility **300** so as to be further processed thereby, e.g., in a secondary processing function, to obtain processed results data that may then be sent back to the local facility **100** for storage **200** thereby. This may be the case where a local practitioner generates sequenced read data using a local data generating resource **100**, e.g., automated sequencer, and then sends that data over the network **30** to a remote computing facility **300**, which then runs one or more functions on that data, such as a Burrows-Wheeler transform or Needlemen-Wunsch and/or Smith-Waterman alignment function on that sequence data, so as to generate results data that may then be transmitted over the internet **30** to the local computing resource **100** so as to be examined thereby in one or more local administered processing protocols and/or stored locally **200**.

[0700] What is needed, however, is a seamless integration between the engagement between local **100** and remote **300** computer processing as well as between local **200** and remote **400** storage, such as in the hybrid cloud **50** based system presented herein. In such an instance, the system can be configured such that local **100** and remote **300** computing resources are configured so as to run seamlessly together, such that data to be processed thereby can be allocated real time to either the local **200** or the remote **300** computing resource without paying an extensive penalty due to transfer rate and/or in operational efficiency. This may be the case, for instance, where the software and/or hardware to be deployed or otherwise run by the computing resources are configured so as to correspond to one another and/or are the same or functionally similar, e.g., the hardware and/or software is configured in the same manner so as to run the same algorithms in the same manner on the generated and/or received data.

[0701] For instance, as can be seen with respect to FIG. **31**A a local computing resource **100** may be configured for generating data, and therefore may include a data generating mechanism **110**, such as for primary data generation and/or analysis, e.g., so as to produce a FASTQ sequence file. This data generating mechanism **110** may be a local computer, as described herein throughout, having a processor that may be configured to run one or more software applications and/or may be hardwired so as to perform one or more algorithms such as in a wired configuration on the generated and/or acquired data. For example, the data generating mechanism **110** may be configured for one or more of generating data, such as sequencing data **111**, which data may be sensed data **111***a*, such as data that is detectable as a change in voltage, ion concentration, electromagnetic radiation, and the like; and/or the data generating mechanism **110** may be configured for generating and/or processing signal, e.g., analog or digital signal data, such as data representing one or more nucleotide identities in a sequence or chain of associated nucleotides. In such an instance, the data generating mechanism **110**, e.g., sequencer **111**, may further be configured for preliminarily processing the generated data so as to perform one or more base call operations **111***c*, such as on the data so as to produce sequence identity data, e.g., a FASTQ file.

[0702] It is to be noted, that in this instance, the data **111** so generated may be generated locally, such as by a local data generating and/or computing resource **150**, e.g., a sequencer on a chip; or it may be produced remotely, e.g., by a remote computing and/or generating resource, such as a remote NGS **300**, but be transferred over the cloud **30**/**50** to the local computing resource **100** such as for secondary processing **150** and/or storage thereby in a local storage resource **200**, such as while awaiting further local processing **150**. In such an instance, where the data generation resource **300** is remote from the local processing **100** and/or storage **200** resources, the corresponding resources may be configured such that the remote and/or local storage, remote and local processing, and/or communicating protocols employed by each resource may be adapted to smoothly and/or seamlessly integrate with one another, e.g., by running the same, similar, and/or equivalent software and/or by having the same, similar, and/or equivalent hardware configurations, and/or employing the same communications and/or transfer protocols, which, in some instances, may have been implemented at the time of manufacture or later thereto.

[0703] Particularly, these functions may be implemented in a hardwired configuration such as where the sequencing function and the secondary processing function are maintained upon the same or associated chip or chipset, e.g., such as where the sequencer and secondary processor are directly interconnected on a chip, as herein described, or may be implemented via software that has been optimized to allow the two remote devices to communicate seamlessly with one another. A combination of optimized hardware and software implementations for performing the recited functions may also be employed, as described herein. In various embodiments, the data generating resource, such as the sequencer **111**, whether implemented in software and/or in hardware or a combination of the same, may further be configured to include an initial tier of processors **500** such as a scheduler, various analytics, comparers, graphers, releasers, and the like, so as to assist the data generator **111**, e.g., sequencer, in converting biological information into raw read data, such as in a FASTQ file format **111***d*.

[0704] Likewise, the same may be true with respect to the performance of the other functions that may be deployed by the local **100** and/or remote **300** computing resources. For example, the local computing resource **100** may include hardware and/or software configured for performing one or more secondary tier **700** of processing functions **112**-**115** on remotely and/or locally generated data, such as genetic sequence data, in a manner that the processing and results thereof may be seamlessly shared with one another and/or stored thereby. Particularly, the local computing function **100** and/or the remote computing function **300** may be configured for generating and/or receiving primary data, such as genetic sequence data, e.g., in a FASTQ or other like file format, and running one or more secondary processing protocols **600** on that generated and/or acquired data, which protocols may be implemented in a software, hardware, or combinational format. For instance, the data generating and/or processing resource **110** may be configured for performing one or more of a mapping operation **112**, an alignment operation **113**, or other related function **114** on the acquired or generated data.

[0705] More particularly, the data generating resource **110** may include a mapping engine **112**, as herein described, or may otherwise include programming for running a mapping algorithm on the genetic sequence data, such as for performing a Burrows-Wheeler transform and/or other algorithms for building a hash table and/or running a hash function **112***a* on said data, such as for hash seed mapping, so as to generate mapped sequence data. The data generating resource **110** may also include an alignment engine **113**, as herein described, or may otherwise include programming for running an alignment algorithm on the genetic sequence data, e.g., mapped sequenced data, such as for performing a gapped and/or gapless Smith-Waterman alignment, and/or Needleman-Wunsch, or other like scoring algorithm **113***a* on said data, so as to generate aligned sequence data. The data generating resource **110** may also be configured to include one or more other modules **114** adapted to perform one or more other processing functions on the genetic sequence data, such as on the mapped and/or aligned sequence data, and thus may include a suitably configured engine **114** or otherwise include programming for running the one or more other processing functions such as a sorting **114***a*, deduplication **114***b*, recalibration **114***c*, local realignment **114***d*, duplicate marking **114***f*, Base Quality Score Recalibration **114***g* function(s) and/or a compression function (such as to produce a BAM, Reduced BAM, and/or a CRAM compression and/or decompression file) **114***e*, in accordance with the methods herein described, which processing functions may be configured as one or more pipelines of the system **1**. Likewise, the system **1** may be configured to include a module **115** adapted for processing the data, e.g., the sequenced, mapped, aligned, and/or sorted data in a manner such as to produce a variant call file **116**, such as in a hardware and/or software based processing functionality. More particularly, the system **1** may include a variant call module **115** for running one or more variant call functions, such as a Hidden Markov Model (HMM) and/or GATK function **115***a* such as in a wired configuration and/or via one or more software applications, e.g., either locally or remotely, and/or a converter **115***b* for the same.

[0706] In particular embodiments, as set forth in FIG. 31B, the system **1** may include a local computing function **100** that may be configured for employing a computer processing resource **150** for performing one or more further computer processing functions on data generated by the system generator **110** or acquired by the system acquisition mechanism **120** (as described below), such as by being transferred thereto, for instance, by a third party **121**, such as via a cloud **30** or hybrid cloud network **50**. For instance, a third party analyzer **121** may deploy a remote computing resource **300** so as to generate relevant data in need of further processing, such as genetic sequence data or the like, which data may be communicated to the system **1** over the network **30**/**50** so as to be further processed. This may be useful, for instance, where the remote computing resource **300** is a NGS, configured for taking raw biological data and converting it to a digital representation thereof, such as in the form of one or more FASTQ files containing reads of genetic sequence data, and where further processing is desired, such as to determine how the generated sequence of an individual differs from that of one or more reference sequences, as herein described, and/or it is desired to subject the results thereof to furthered, e.g., tertiary, processing.

[0707] In such an instance, the system **1** may be adapted so as to allow one or more parties, e.g., a primary and/or secondary and/or third party user, to access the associated local processing resources **100**, and/or a suitably configured remote processing resource **300** associated therewith, in a manner so as to allow the user to perform one or more quantitative and/or qualitative processing functions **152** on the generated and/or acquired data. For instance, in one configuration, the system **1** may include, e.g., in addition to primary **600** and/or secondary **600** processing pipelines, a third tier of processing modules **700**, which processing modules may be configured for performing one or more processing functions on the generated and/or acquired primary and/or secondary processed data.

[0708] Particularly, in one embodiment, the system **1** may be configured for generating and/or receiving processed genetic sequence data **111** that has been either remotely or locally mapped **112**, aligned **113**, sorted **114***a*, and/or further processed **114** so as to generate a variant call file **116**, which variant call file may then be subjected to further processing such as within the system **1**, such as in response to a second and/or third party analytics requests **121**. More particularly, the system **1** may be configured to receive processing requests from a third party **121**, and further be configured for performing such requested tertiary processing **700** on the generated and/or acquired data. Specifically, the system **1** may be configured for producing and/or acquiring genetic sequence data **111**, may be configured for taking that genetic sequence data and mapping **112**, aligning **113**, and/or sorting **114***a* it to produce one or more variant call files (VCFs) **116**, and additionally the system **1** may be configured for performing a tertiary processing function **700** on the data, e.g., with respect to the one or more VCFs. The system **1** may be configured so as to perform any form of tertiary processing **700** on the generated and/or acquired data, such as by subjecting it to one or more pipeline processing functions **700** such as to generate genome data **122***a*, epigenome data **122***b*, metagenome data **122***c*, and the like, including joint genotyping **122***d*, GATK **122***e* and/or MuTect2 **122***f* analysis pipelines. Further, the system **1** may be configured for performing an additional tier of processing on the generated and/or processed data, such as including one or more of non-invasive prenatal testing (NIPT) **123***a*, N/P ICU **123***b*, cancer related diagnostics and/or therapeutic modalities **123***c*, various laboratory developed tests (LDT) **123***d*, agricultural biological (Ag Bio) applications **123***e*, or other such health care related **123***f* processing function.

[0709] Hence, in various embodiments, where a primary user may access and/or configure the system **1** and its various components directly, such as through direct access therewith, such as through the local computing resource **100**, as presented herein, the system **1** may also be adapted for being accessed by a secondary party, such as is connected to the system **1** via a local network or intranet connection **10** so as to configure and run the system **1** within the local environment. Additionally, in certain embodiments, as presented in FIG. **31**B, the system may be adapted for being accessed and/or configured by a third party **121**, such as over an associated hybrid-cloud network **50** connecting the third party **121** to the system **1**, such as through an application program interface (API), accessible as through one or more graphical user interface (GUI) components. Such a GUI may be configured to allow the third party user to access the system **1**, and using the API configure the various components of the system, the modules, associated pipelines, and other associated data generating and/or processing functionalities so as to run only those system components necessary and/or useful to the third party and/or requested or desired to be run thereby.

[0710] Accordingly, in various instances, the system **1** as herein presented may be adapted so as to be configurable by a primary, secondary, or tertiary user of the system. In such an instance, the system **1** may be adapted to allow the user to configure the system **1** and thereby to arrange its components in such a manner as to deploy one, all, or a selection of the analytical system resources, e.g., **152**, to be run on data that is either generated, acquired, or otherwise transferred to the system, e.g., by the primary, secondary, or third party user, such that the system **1** runs only those portions of the system necessary or useful for running the analytics requested by the user to obtain the desired results thereof. For example, for these and other such purposes, an API may be included within the system **1** wherein the API is configured so as to include or otherwise be operably associated with a graphical user interface (GUI) including an operable menu and/or a related list of system function calls from which the user can select and/or otherwise make so as to configure and operate the system and its components as desired.

[0711] In such an instance, the GUI menu and/or system function calls may direct the user selectable operations of one or more of a first tier of operations **600** including: sequencing **111**, mapping **112**, aligning **113**, sorting **114***a*, variant calling **115**, and/or other associated functions **114** in accordance with the teachings herein, such as with relation to the primary and/or secondary processing functions herein described. Further, where desired the GUI menu and/or system function calls may direct the operations of one or more of a second tier of operations **700** including: a genome pipeline **122***a*, epigenome pipeline **122***b*, metagenome pipeline **122***c*, a joint genotyping pipeline **122***d*, GATK **122***e* and/or MuTect2 **122***f* analysis pipelines. Furthermore, where desired the GUI menu and system function calls may direct the user selectable operations of one or more of a third tier of operations **800** including: non-invasive prenatal testing (NIPT) **123***a*, N/P ICU **123***b*, cancer related diagnostics and/or therapeutic modalities **123***c*, various laboratory developed tests (LDT) **123***d*, agricultural biological (Ag Bio) applications **123***e*, or other such health care related **123***f* processing functions.

[0712] Accordingly, the menu and system function calls may include one or more primary, secondary, and/or tertiary processing functions, so as to allow the system and/or its component parts to be configured such as with respect to performing one or more data analysis pipelines as selected and configured by the user. In such an instance, the local computing resource **100** may be configured to correspond to and/or mirror the remote computing resource **300**, and/or likewise the local storage resource **200** may be configured to correspond and/or mirror the remote storage resource **400** so that the various components of the system may be run and/or the data generated thereby may be stored either locally or remotely in a seamless distributed manner as chosen by the use of the system **1**. Additionally, in particular embodiments, the system **1** may be made accessible to third parties, for running proprietary analysis protocols **121***a* on the generated and/or processed data, such as by running through an artificial intelligence interface designed to find correlations there between.

[0713] The system **1** may be configured so as to perform any form of tertiary processing on the generated and/or acquired data. Hence, in various embodiments, a primary, secondary, or tertiary user may access and/or configure any level of the system **1** and its various components either directly, such as through direct access with the computing resource **100**, indirectly, such as via a local network connection **10**, or over an associated hybrid-cloud network **50** connecting the party to the system **1**, such as through an appropriately configured API having the appropriate permissions. In such an instance, the system components may be presented as a menu, such as a GUI selectable menu, where the user can select from all the various processing and storage options desired to be run on the user presented data. Further, in various instances, the user may upload their own system protocols so as to be adopted and run by the system so as to process various data in a manner designed and selected for by the user. In such an instance, the GUI and associated API will allow the user to access the system **1** and using the API add to and configure the various components of the system, the modules, associated pipelines, and other associated data generating and/or processing functionalities so as to run only those system components necessary and/or useful to the party and/or requested or desired to be run thereby.

[0714] Where the above with respect to FIGS. **31**A and **31**B are directed to data generation **110** such as local data generation **100**, employing a local computing resource **150**; as indicated above, and with respect to FIG. **31**C, one or more of the above demarcated modules, and their respective functions and/or associated resources, may be configured for being performed remotely, such as by a remote computing resource **300**, and further be adapted to be transmitted to the system **1**, such as in a seamless transfer protocol over a cloud based internet connection **30**/**50**, such as via a suitably configured data acquisition mechanism **120**.

[0715] Accordingly, in such an instance, the local computing resource **100** may include a data acquisition mechanism **120**, such as configured for transmitting and/or receiving such acquired data and/or associated information. For instance, the system **1** may include a data acquisition mechanism **120** that is configured in a manner so as to allow the continued processing and/or storage of data to take place in a seamless and steady manner, such as over a cloud or hybrid based network **30**/**50** where the processing functions are distributed both locally **100** and/or remotely **300**, and likewise where one or more of the results of such processing may be stored locally **200** and/or remotely **400**, such that the system seamlessly allocates to which local or remote resource a given job is to be sent for processing and/or storage regardless of where the resource is physically positioned. Such distributed processing, transferring, and acquisition may include one or more of sequencing **111**, mapping **112**, aligning **113**, sorting **114***a*, duplicate marking **114***c*, deduplication, recalibration **114***d*, local realignment **114***e*, Base Quality Score Recalibration **114***f* function(s) and/or a compression function **114***g*, as well as a variant call function **116**, as herein described. Where stored locally **200** or remotely **400**, the processed data, in whatever state it is in the process may be made available to either the local **100** or remote processing **300** resources, such as for further processing prior to re-transmission and/or re-storage.

[0716] Specifically, the system **1** may be configured for producing and/or acquiring genetic sequence data **111**, may be configured for taking that genetic sequence data **111** and processing it locally **150**, or transferring the data over a suitably configured cloud **30** or hybrid cloud **50** network such as to a remote processing facility for remote processing **300**. Further, once processed the system **1** may be configured for storing the processed data remotely **400** or transferring it back for local storage **200**. Accordingly, the system **1** may be configured for either local or remote generation and/or processing of data, such as where the generation and/or processing steps may be from a first tier of primary and/or secondary processing functions **600**, which tier may include one or more of: sequencing **111**, mapping **112**, aligning **113**, and/or sorting **114***a* so as to produce one or more variant call files (VCFs) **116**. Likewise, the system **1** may be configured for either local or remote generation and/or processing of data, such as where the generation and/or processing steps may be from a second tier of tertiary processing functions **700**, which tier may include one or more of generating and/or acquiring data pursuant to a genome pipeline **122***a*, epigenome pipeline **122***b*, metagenome pipeline **122***c*, a joint genotyping pipeline **122***d*, GATK **122***e* and/or MuTect2 **122***f* analysis pipeline. Additionally, the system **1** may be configured for either local or remote generation and/or processing of data, such as where the generation and/or processing steps may be from a third tier of tertiary processing functions **800**, which tier may include one or more of generating and/or acquiring data related to and including: non-invasive prenatal testing (NIPT) **123***a*, N/P ICU **123***b*, cancer related diagnostics and/or therapeutic modalities **123***c*, various laboratory developed tests (LDT) **123***d*, agricultural biological (Ag Bio) applications **123***e*, or other such health care related **123***f* processing functions.

[0717] In particular embodiments, as set forth in FIG. **31**C, the system **1** may further be configured for allowing one or more parties to access the system and transfer information to or from the associated local processing **100** and/or remote **300** processing resources as well as to store information either locally **200** or remotely **400** in a manner that allows the user to choose what information get processed and/or stored where on the system **1**. In such an instance, a user can not only decide what primary, secondary, and/or tertiary processing functions get performed on generated and/or acquired data, but also how those resources get deployed, and/or where the results of such processing gets stored. For instance, in one configuration, the user may select whether data is generated either locally or remotely, or a combination thereof, whether it is subjected to secondary processing, and if so, which modules of secondary processing it is subjected to, and/or which resource runs which of those processes, and further may determine whether the then generated or acquired data is further subjected to tertiary processing, and if so, which modules and/or which tiers of tertiary processing it is subjected to, and/or which resource runs which of those processes, and likewise, where the results of those processes are stored for each step of the operations.

[0718] Particularly, in one embodiment, the user may configure the system **1** of FIG. **31**A so that the generating of genetic sequence data **111** takes place remotely, such as by an NGS, but the secondary processing **600** of the data occurs locally **100**. In such an instance, the user can then determine which of the secondary processing functions occur locally **100**, such as by selecting the processing functions, such as mapping **112**, aligning **113**, sorting **111**, and/or producing a VCF **116**, from a menu of available processing options. The user may then select whether the locally processed data is subjected to tertiary processing, and if so which modules are activated so as to further process the data, and whether such tertiary processing occurs locally **100** or remotely **300**. Likewise, the user can select various options for the various tiers of tertiary processing options, and where any generated and/or acquired data is to be stored, either locally **200** or remotely **400**, at any given step or time of operation.

[0719] More particularly, a primary user may configure the system to receive processing requests from a third party, where the third party may configure the system so as for performing such requested primary, secondary, and/or tertiary processing on generated and/or acquired data. Specifically, the user or second or third party may configure the system **1** for producing and/or acquiring genetic sequence data, either locally **100** or remotely **200**, may configure the system **1** for taking that genetic sequence data and mapping, aligning, and/or sorting it, either locally or remotely, so as to produce one or more variant call files (VCFs), and additionally may configure the system for performing a tertiary processing function on the data, e.g., with respect to the one or more VCFs, either locally or remotely. More particular still, the user or other party may configure the system **1** so as to perform any form of tertiary processing on the generated and/or acquired data, and where that processing is to occur in the system. Hence, in various embodiments, the first, second, and/or third party **121** user may access and/or configure the system **1** and its various components directly such as by directly accessing the local computing function **100**, via a local network connection **10**, or over an associated hybrid-cloud network **50** connecting the party **121** to the system **1**, such as through an application program interface (API), accessible as through one or more graphical user interface (GUI) components. In such an instance, the third party user may access the system **1** and use the API to configure the various components of the system, the modules, associated pipelines, and other associated data generating and/or processing functionalities so as to run only those system components necessary and/or useful to the third party and/or requested or desired to be run thereby, and further allocate which computing resources will provide the requested processing, and where the results data will be stored.

[0720] Accordingly, in various instances, the system **1** may be configurable by a primary, secondary, or tertiary user of the system who can configure the system **1** so as to arrange its components in such a manner as to deploy one, all, or a selection of the analytical system resources to be run on data that the user either directly generates, causes to be generated by the system **1**, or causes to be transferred to the system **1**, such as over a network associated therewith, such as via the data acquisition mechanism **120**. In such a manner, the system **1** is configurable so as to only run those portions of the system necessary or useful for the analytics desired and/or requested by the requesting party. For example, for these and other such purposes, an API may be included wherein the API is configured so as to include a GUI operable menu and/or a related list of system function calls that from which the user can select so as to configure and operate the system as desired. Additionally, in particular embodiments, the system **1** may be made accessible to third parties, such as governmental regulators, such as the Federal Drug Administration (FDA) **70***b*, or allow third parties to collate, compile, and/or access a data base of genetic information derived or otherwise acquired and/or compiled by the system **1** so as to form an electronic medical records (EMR) database **70***a* and/or to allow governmental access and/or oversight of the system, such as the FDA for Drug Development Evaluation. The system **1** may also be set up to conglomerate, compile, and/or annotate the data **70***c* and/or allow other high level users access thereto.

[0721] Accordingly, in various embodiments, as can be seen with respect to FIG. **32**A, a hybrid cloud **50** is provided wherein the hybrid cloud is configured for connecting a local computing **100** and/or storage resource **200** with a remote computing **300** and/or storage **400** resource, such as where the local and remote resources are separated one from the other distally, spatially, geographically, and the like. In such an instance, the local and distal resources may be configured for communicating with one another in a manner so as to share information, such as digital data, seamlessly between the two. Particularly, the local resources may be configured for performing one or more types of processing on the data, such as prior to transmission across the hybrid network **50**, and the remote resources may be configured for performing one or more types of further processing of the data.

[0722] For instance, in one particular configuration, the system **1** may be configured such that a generating and/or analyzing function **152** is configured for being performed locally **100** by a local computing resource, such as for the purpose of performing a primary and/or secondary processing function, so as to generate and/or process genetic sequence data, as herein described. Additionally, in various embodiments, the local resources may be configured for performing one or more tertiary processing functions on the data, such as one or more of genome, exome, and/or epigenome analysis, or a cancer, microbiome, and/or other DNA/RNA processing analysis. Further, where such processed data is meant to be transferred, such as to a remote computing **300** and/or storage **400** resource, the data may be transformed such as by a suitably configured transformer **151**, which transformer **151** may be configured for indexing, converting, compressing, and/or encrypting the data, such as prior to transfer over the hybrid network **50**.

[0723] In particular instances, such as where the generated and processed data is transferred to a remote computing resource **300** for further processing, such processing may be of a global nature and may include receiving data from a plurality of local computing resources **100**, collating such pluralities of data, annotating the data, and comparing the same, such as to interpret the data, determine trends thereof, analyzing the same for various biomarkers, and aiding in the development of diagnostics, therapeutics, and/or prophylactics. Accordingly, in various instances, the remote computing resource **300** may be configured as a data processing hub, such as where data from a variety of sources may be transferred, processed, and/or stored while waiting to be transformed and/or transferred, such as by being accessed by the local computing resource **100**. More particularly, the remote processing hub **300** may be configured for receiving data from a plurality of resources **100**, processing the same, and distributing the processed data back to the variety of local resources **100** so as to allow for collaboration amongst researchers and/or resources **100**. Such collaboration may include various data sharing protocols, and may additionally include preparing the data to be transferred, such as by allowing a user of the system **1** to select amongst various security protocols and/or privacy settings so as to control how the data will be prepared for transfer.

[0724] In one particular instance, as presented in FIG. **32**B, a local computing **100** and/or storage **200** resource is provided, such as on-site at a user's location. The computing resource **100** and/or storage **200** resource may be coupled to a data generating resource **121**, such as an NGS or sequencer on a chip, as herein described, such as over a direct or an intranet connection **10**, where the sequencer **121** is configured for generating genetic sequencing data, such as FASTQ files. For instance, the sequencer **121** may be part of and/or housed in the same apparatus as that of the computing resource **100** and/or storage unit **200**, so as to have a direct communicable and/or operable connection therewith, or the sequencer **121** and computing resource **100** and/or storage resource **200** may be part of separate apparatuses from one another, but housed in the same facility, and thus connected over a cabled or intranet **10** connection. In some instances, the sequencer **121** may be housed in a separate facility than that of the computing **100** and/or storage **200** resource and thus may be connected over an internet **30** or hybrid cloud connection **50**.

[0725] In such instances, the genetic sequence data may be processed **100** and stored locally **200**, prior to being transformed, by a suitably configured transformer **151**, or the generated sequence data may be transmitted directly to one or more of the transformer **151** and/or analyzer **152**, such as over a suitably configured local connection **10**, intranet **30**, or hybrid cloud connection **50**, as described above such as prior to being processed locally. Particularly, like the data generating resource **121**, the transformer **151** and/or analyzer **152** may be part of and/or housed in the same apparatus as that of the computing resource **100** and/or storage unit **200**, so as to have a direct communicable and/or operable connection therewith, or the transformer **151** and/or analyzer **152** and computing resource **100** and/or storage resource **200** may be part of separate apparatuses from one another, but housed in the same facility, and thus connected over a cabled or intranet **10** connection. In some instances, the transformer **151** and/or analyzer **152** may be housed in a separate facility than that of the computing **100** and/or storage **200** resource and thus may be connected over an internet **30** or hybrid cloud connection **50**.

[0726] In such instances, the transformer **151** may be configured for preparing the data to be transmitted either prior to analysis or post analysis, such as by a suitably configured computing resource **100** and/or analyzer **152**. For instance, the analyzer **152** may perform a secondary and/or tertiary processing function on the data, as herein described, such as for analyzing the generated sequence data with respect to determining its genomic and/or exomic characteristics **152***a*, its epigenomic features **152***b*, any various DNA and/or RNA markers of interests and/or indicators of cancer **152***c*, and its relationships to one or more microbiomes **152***d*, as well as one or more other secondary and/or tertiary processes as described herein. As indicated, the generated and/or processed data may be transformed, such as by a suitably configured transformer **151** such as prior to transmission throughout the system **1** from one component thereof to another, such as over a direct, local **10**, internet **30**, or hybrid cloud **50** connection. Such transformation may include one or more of conversion **151***d*, such as where the data is converted from one form to another; comprehension **151***c*, including the coding, decoding, and/or otherwise taking data from an incomprehensible form and transforming it to a comprehensible form, or from one comprehensible form to another; indexing **151***b*, such as including compiling and/or collating the generated data from one or more resources, and making it locatable and/or searchable, such as via a generated index; and/or encryption **151***a*, such as creating a lockable and unlockable, password protected dataset, such as prior to transmission over an internet **30** and/or hybrid cloud **50**.

[0727] Hence, in these and/other such instances, the hybrid cloud **50** may be configured for allowing seamless and protected transmission of data throughout the components of the system, such as where the hybrid cloud **50** is adapted to allow the various users of the system to configure its component parts and/or the system itself so as to meet the research, diagnostic, therapeutic and/or prophylactic discovery and/or development needs of the user. Particularly, the hybrid cloud **50** and/or the various components of the system **1** may be operably connected with compatible and/or corresponding API interfaces that are adapted to allow a user to remotely configure the various components of the system **1** so as to deploy the resources desired in the manner desired, and further to do so either locally, remotely, or a combination of the same, such as based on the demands of the system and the particulars of the analyses being performed, all the while being enabled to communicate in a secured, encryptable environment. Another exemplary embodiment of the hybrid cloud system, as herein presented, is depicted in FIG. **32**C.

[0728] For instance, as can be seen with respect to FIG. **33**, the system **1** may be a multi-tiered and/or multiplexed bioanalytical processing platform that includes layers of processing units each having one or more processing pipelines that may be deployed in a systematic and/or concurrent and/or sequential manner so as to process genetic information from its primary processing stage **400**/**500**, so as to produce genetic sequence data, such as in one or more FASTQ files; to its secondary processing stage **600**, so as to produce one or more variant call files; and further to take the one or more variant call files, or other associated processed data, and perform one or more other operations thereon such as for the purposes of performing one or more diagnostics and/or prophylactic and/or therapeutic procedures there with, such as in response to a third party request **121** and/or in response to data submitted by the third party **121**. Such further processing may include various pipeline protocols **700**, such as configured so as to run analytics on the determined genetic variation data of one or more subjects, including genome, epigenome, metagenome, and/or genotyping analytics, such as in one tier, and/or various disease diagnostic and/or research protocols **800**, which may include one or more of NIPT, NICU, cancer, LDT, biological, AgBio applications and the like. Particularly, the system **1** may further be adapted so as to receive and/or transmit various data **900** related to the procedures and processes herein such as related to electronic medical records (EMR) data, Federal Drug Administration testing and/or structuring data, data relevant to annotation, and the like. Such data may be useful so as to allow a user to make and/or allow access to generated medical, diagnostic, therapeutic, and/or prophylactic modalities developed through use of the system **1** and/or made accessible thereby.

[0729] Hence, one or more, e.g., all, of these functions therefore may be performed locally, e.g., on site **10**, on the cloud **30**, or via controlled access through the hybrid cloud **50**. In such an instance, developer environment is created that allows the user to control the functionality of the system to meet his or her individual needs and/or to allow access thereto for others seeking the same or similar results. Consequently, the various components, processes, procedures, tools, tiers, and hierarchies of the system may be configurable such as via a GUI interface that allows the user to select which components to be run on which data at what time in what order in accordance with the user determined desires and protocols so as to generate relevant data and connections between data that may be securely communicated throughout the system whether locally or remotely. As indicated, these components can be made to communicate seamlessly together regardless of location and/or how connected, such as by being configurable so as to run the same or similar processes in the same or similar manner such as by employing corresponding API interfaces dispersed throughout the system the employment of which allows the various users to configure the various components to run the various procedures in like manner.

[0730] For instance, an API may be defined in a header file with respect to the processes to be run by each particular component of the system **1**, wherein the header describes the functionality and determines how to call a function, such as the parameters that are passed, the inputs received and outputs transmitted, and the manner in which this occurs, what comes in and how, what goes out and how, and what gets returned, and in what manner. For example, in various embodiments, one or more of the components and/or elements thereof, which may form one or more pipelines of one or more tiers of the system may be configurable such as by instructions entered by a user and/or one or more second and/or third party applications. These instructions may be communicated to the system via the corresponding APIs which communicate with one or more of the various drivers of the system, instructing the driver(s) as to which parts of the system, e.g., which modules and/or which processes thereof are to be activated, when, and in what order, given a preselected parameter configuration, which may be determined by a user selectable interface, e.g., GUI.

[0731] As described above, the one or more DMA drivers of the system **1** may be configured to run in corresponding fashion, such as at the kernel level of each component and the system **1** as a whole. In such an instance, one or more of the provided kernel's may have their own very low level, basic API that provides access to the hardware and functions of the various components of the system **1** so as to access applicable registers and modules so as to configure and direct the processes and the manners in which they are run on the system **1**. Particularly, on top of this layer, a virtual layer of service functions may be built so as to form the building blocks that are used for a multiplicity of functions that send files down to the kernel(s) and get results back, encodes, encrypts, and/or transmits the relevant data and further performs more higher level functions thereon. On top of that layer an additional layer may be built that uses those service functions, which may be an API level that a user may interface with, which may be adapted to function primarily for configuration of the system **1** as a whole or its component parts, downloading files, and uploading results, which files and/or results may be transmitted throughout the system either locally or globally.

[0732] Such configuration may include communicating with registers and also performing function calls. For example, as described herein above, one or more function calls necessary and/or useful to perform the steps, e.g., sequentially, to execute a mapping and/or aligning and/or sorting and/or variant call, or other secondary and/or tertiary function as herein described may be implemented in accordance with the hardware operations and/or related algorithms so as to generate the necessary processes and perform the required steps.

[0733] Specifically, because in certain embodiments one or more of these operations may be based on one or more structures, the various structures needed for implementing these operations may need to be constructed. There will therefore be a function call that performs this function, which function call will cause the requisite structure to be built for the performance of the operation, and because of this a call will accept a file name of where the structure parameter files are stored and will then generate one or more data files that contain and/or configure the requisite structure. Another function call may be to load the structure that was generated via the respective algorithm and transfer that down to the memory on the chip and/or system **1**, and/or put it at the right spot where the hardware is expecting them to be. Of course, various data will need to be downloaded onto the chip and/or otherwise be transferred to the system generator, as well for the performance of the various other selected functions of the system **1**, and the configuration manager can perform these functions, such as by loading everything that needs to be there in order for the modules of pipelines of the tiers of the platforms of the chip and/or system as a whole to perform their functions, into a memory on, attached, or otherwise associated with the chip and/or system.

[0734] Additionally, the API may be configured to allow one or more chips of the system **1** to interface with the circuit board of the sequencer **121**, the computing resource **100**/**300**, transformer **151**, analyzer **152**, interpreter **310**, collaborator **320**, or other system component, when included therewith, so as to receive the FASTQ and/or other generated and/or processed genetic sequencing files directly from the sequencer or other processing component such as immediately once they have been generated and/or processed and then transfers that information to the configuration manager which then directs that information to the appropriate memory banks in the hardware and/or software that makes that information available to the pertinent modules of the hardware, software, and/or system as a whole so that they can perform their designated functions on that information so as to call bases, map, align, sort, etc. the sample DNA/RNA with respect to the reference genome, and or to run associated secondary and/or tertiary processing operations thereon.

[0735] Accordingly, in various embodiments, a client level interface (CLI) may be included wherein the CLI may allow the user to call one or more of these functions directly. In various embodiments, the CLI may be a software application, e.g., having a GUI, that is adapted to configure the accessibility and/or use of the hardware and/or various other software applications of the system. The CLI, therefore, may be a program that accepts instructions, e.g., arguments, and makes functionality available simply by calling an application program. As indicated above, the CLI can be command line based or GUI (graphical user interface) based. The line based commands happen at a level below the GUI, where the GUI includes a windows based file manager with click on function boxes that delineate which modules, which pipelines, which tiers, of which platforms will be used and the parameters of their use. For example, in operation, if instructed, the CLI will locate the reference, will determine if a hash table and/or index needs to be generated, or if already generated locate where it is stored, and direct the uploading of the generated hash table and/or index, etc. These types of instructions may appear as user options at the GUI that the user can select the associated chip(s)/system **1** to perform.

[0736] Furthermore, a library may be included wherein the library may include pre-existing, editable, configuration files, such as files orientated to the typical user selected functioning of the hardware and/or associated software, such as with respect to a portion or whole genome and/or protein analysis, for instance, for various analyses, such as personal medical histories and ancestry analysis, or disease diagnostics, or drug discovery, therapeutics, and/or one or more of the other analytics, etc. These types of parameters may be preset, such as for performing such analyses, and may be stored in the library. For example, if the platform herein described is employed such as for NIPT, NICU, Cancer, LDT, AgBio, and related research on a collective level, the preset parameters may be configured differently than if the platform were directed simply to researching genomic and/or genealogy based research, such as on an individual level.

[0737] More particularly, for specific diagnosis of an individual, accuracy may be an important factor, therefore, the parameters of the system may be set to ensure increased accuracy albeit in exchange for possibly a decrease in speed. However, for other genomics applications, speed may be the key determinant and therefore the parameters of the system may be set to maximize speed, which however may sacrifice some accuracy. Accordingly, in various embodiments, often used parameter settings for performing different tasks can be preset into the library to facilitate ease of use. Such parameter settings may also include the necessary software applications and/or hardware configurations employed in running the system **1**. For instance, the library may contain the code that executes the API, and may further include sample files, scripts, and any other ancillary information necessary for running the system **1**. Hence, the library may be configured for compiling software for running the API as well as various of the executables.

[0738] Additionally, as can be seen with respect to FIG. **32**C, the system may be configured such that one or more of the system components may be performed remotely, such as where the system component is adapted to run one or more comparative functions on the data, such as an interpretive function **310** and/or collaborative function **320**. For instance, where an interpretive protocol is employed on the data, the interpretive protocol **312** may be configured to analyze and draw conclusions about the data and/or determine various relationships with respect thereto, one or more other analytical protocols may also be performed and include annotating the data **311**, performing a diagnostic **313** on the data, and/or analyzes the data, so as to determine the presence or absence of one or more biomarkers **314**. Additionally, where a collaborative protocol is performed, the system **1** may be configured for providing an electronic forum where data sharing **321** may occur, which data sharing protocol may include user selectable security **324** and/or privacy **322** settings that allow the data to be encrypted and/or password protected, so that the identity and sources of the data may be hidden from a user of the system **1**. In particular instances, the system **1** may be configured so as to allow a 3rd party analyzer **121** to run virtual simulations on the data. Further, one generated, the interpreted data and/or the data subjected to one or more collaborative analyses may be stored either remotely **400** or locally **200** so as to be made available to the remote **300** or local **100** computing resources, such as for further processing and/or analysis.

[0739] In another aspect, as can be seen with respect to FIG. **34**, a method for using the system to generate one or more data files upon which one or more secondary and/or tertiary processing protocols may be run is provided. For instance, the method may include providing a genomic infrastructure such as for one or more of onsite, cloud-based, and/or hybrid genomic and/or bioinformatics generation and/or processing and/or analysis.

[0740] In such an instance, the genomic infrastructure may include a bioinformatics processing platform having one or more memories that are configured to store one or more configurable processing structures for configuring the system so as to be able to perform one or more analytical processing functions on data, such as data including a genomic sequence of interest or processed result data pertaining thereto. The memory may include the genomic sequence of interest to be processed, e.g., once generated and/or acquired, one or more genetic reference sequences, and/or may additionally include an index of the one or more genetic reference sequences and/or a list of splice junctions pertaining thereto. The system may also include an input having a platform application programming interface (API) for selecting from a list of options one or more of the configurable processing structures, such as for configuring the system, such as by selecting which processing functions of the system will be run on the data, e.g., the pre- or processed genomic sequences of interest. A graphical user interface (GUI) may also be present, such as operably associated with the API, so as to present a menu by which a user can select which of the available options he or she desires to be run on the data.

[0741] The system may be implemented on one or more integrated circuits that may be formed of one or more sets of configurable, e.g., preconfigured and/or hardwired, digital logic circuits that may be interconnected by a plurality of physical electrical interconnects. In such an instance, the integrated circuit may have an input, such as a memory interface, for receiving one or a plurality of the configurable structure protocols, e.g., from the memory, and may further be adapted for implementing the one or more structures on the integrated circuit in accordance with the configurable processing structure protocols. The memory interface of the input may also be configured for receiving the genomic sequence data, which may be in the form of a plurality of reads of genomic data. The interface may also be adapted for accessing the one or more genetic reference sequences and the index(es).

[0742] In various instances, the digital logic circuits may be arranged as a set of processing engines that are each formed of a subset of the digital logic circuits. The digital logic circuits and/or processing engines may be configured so as to perform one or more pre-configurable steps of a primary, secondary, and/or tertiary processing protocol so as to generate the plurality of reads of genomic sequence data, and/or for processing the plurality of reads of genomic data, such as according to the genetic reference sequence(s) or other genetic sequence derived information. The integrated circuit may further have an output so as to output result data from the primary, secondary, and/or tertiary processing, such as according to the platform application programming interface (API).

[0743] Particularly, in various embodiments, the digital logic circuits and/or the sets of processing engines may form a plurality of genomic processing pipelines, such as where each pipeline may have an input that is defined according to the platform application programming interface so as to receive the result data from the primary and/or secondary processing by the bioinformatics processing platform, and for performing one or more analytic processes thereon so as to produce result data. Additionally, the plurality of genomic processing pipelines may have a common pipeline API that defines a secondary and/or tertiary processing operation to be run on the result data from the primary and/or secondary processed data, such as where each of the plurality of genomic processing pipelines is configured to perform a subset of the secondary and/or tertiary processing operations and to output result data of the secondary and/or tertiary processing according to the pipeline API.

[0744] In such instances, a plurality of the genomic analysis applications may be stored in the memory and/or an associated searchable application repository, such as where each of the plurality of genomic analysis applications are accessible via an electronic medium by a computer such as for execution by a computer processor, so as to perform a targeted analysis of the genomic pre- or post processed data from the result data of the primary, secondary, and/or tertiary processing, such as by one or more of the plurality of genomic processing pipelines. In particular instances, each of the plurality of genomic analysis applications may be defined by the API and may be configured for receiving the result data of the primary, secondary, and/or tertiary processing, and/or for performing the target analysis of the pre- or post processed genomic data, and for outputting the result data from the targeted analysis to one of one or more genomic databases.

[0745] The method may additionally include, selecting, e.g., from the menu of the GUI, one or more genomic processing pipelines from a plurality of the available genomic processing pipelines of the system; selecting one or more genomic analysis applications from the plurality of genomic analysis applications that are stored in an application repository; and executing, using a computer processor, the one or more selected genomic analysis applications to perform a targeted analysis of genomic data from the result data of the primary, secondary, and/or tertiary processing.

[0746] Additionally, in various embodiments, all of mapping, aligning, and sorting, may take place on the chip, and local realignment, duplicate marking, base quality score recalibration may, and/or one or more of the tertiary processing protocols and/or pipelines, in various embodiments, also take place on the chip, and in various instances, various compression protocols, such as BAM and CRAM, may also take place on the chip. However, once the primary, secondary, and/or tertiary processed data has been produced, it may be compressed, such as prior to being transmitted, such as by being sent across the system, being sent up to the cloud, such as for the performance of the variant calling module, a secondary, tertiary, and/or other processing platform, such as including an interpretive and/or collaborative analysis protocol. This might be useful especially given the fact that variant calling, including the tertiary processing thereof, can be a moving target, e.g., there is not one standardized agreed upon algorithm that the industry uses.

[0747] Hence, different algorithms can be employed, such as by remote users, so as to achieve a different type of result, as desired, and as such having a cloud based module for the performance of this function may be useful for allowing the flexibility to select which algorithm is useful at any particular given moment, and also as for serial and/or parallel processing. Accordingly, any one of the modules disclosed herein can be implemented as either hardware, e.g., on the chip, or software, e.g., on the cloud, but in certain embodiments, all of the modules may be configured so that their function may be performed on the chip, or all of the modules may be configured so that their function may be performed remotely, such as on the cloud, or there will be a mixture of modules wherein some are positioned on one or more chips and some are positioned on the cloud. Further, as indicated, in various embodiments, the chip(s) itself may be configured so as to function in conjunction with, and in some embodiments, in immediate operation with a genetic sequencer, such as an NGS and/or sequencer on a chip.

[0748] More specifically, in various embodiments, an apparatus of the disclosure may be a chip, such as a chip that is configured for processing genomics data, such as by employing a pipeline of data analysis modules. According, as can be seen with respect to FIG. **36**, a genomics pipeline processor chip **100** is provided along with associated hardware of a genomics pipeline processor system **10**. The chip **100** has one or more connections to external memory **102** (at "DDR3 Mem Controller"), and a connection **104** (e.g., "PCIe Interface") to the outside world, such as a host computer **106**, for example. A crossbar **108** (e.g., switch) provides access to the memory interfaces to various requestors. DMA engines **110** transfer data at high speeds between the host and the processor chip's **100** external memories **102** (via the crossbar **108**), and/or between the host and a central controller **112**. The central controller **112** controls chip operations, especially coordinating the efforts of multiple processing engines. The processing engines are formed of a set of hardwired digital logic circuits that are interconnected by physical electrical interconnects, and are organized into engine clusters **114**. In some implementations, the engines in one cluster share one crossbar port, via an arbiter. The central controller **112** has connections to each of the engine clusters. Each engine cluster **114** has a number of processing engines for processing genomic data, including a mapper **120** (or mapping module), an aligner **122** (or aligning module), and a sorter **124** (or sorting module). An engine cluster **114** can include other engines or modules, such as a variant caller module, as well.

[0749] In accordance with one data flow model consistent with implementations described herein, the host sends commands and data via the DMA engines **110** to the central controller **112**, which load-balances the data to the processing engines. The processing engines return processed data to the central controller **112**, which streams it back to the host via the DMA engines **110**. This data flow model is suited for mapping and alignment.

[0750] In accordance with an alternative data flow model consistent with implementations described herein, the host streams data into the external memory, either directly via DMA engines **110** and the crossbar **108**, or via the central controller **112**. The host sends commands to the central controller **112**, which sends commands to the processing engines, which instruct the processing engines as to what data to process. The processing engines access input data from the external memory, process it, and write results back to the external memory, reporting status to the central controller **112**. The central controller **112** either streams the result data back to the host from the external memory, or notifies the host to fetch the result data itself via the DMA engines **110**.

[0751] FIG. **37** illustrates a genomics pipeline processor system **20**, showing a full complement of processing engines inside an engine cluster **214**. The pipeline processor system **20** may include one or more engine clusters **214**. In some implementations, the pipeline processor system **20** includes four our more engine clusters **214**. The processing engines or processing engine types can include, without limitation, a mapper, an aligner, a sorter, a local realigner, a base quality recalibrater, a duplicate marker, a variant caller, a compressor and/or a decompressor. In some implementations, each engine cluster **214** has one of each processing engine type. Accordingly, all processing engines of the same type can access the crossbar **208** simultaneously, through different crossbar ports, because they are each in a different engine cluster **214**. Not every processing engine type needs to be formed in every engine cluster **214**. Processing engine types that require massive parallel processing or memory bandwidth, such as the mapper (and attached aligner(s)) and sorter, may appear in every engine cluster of the pipeline processor system **20**. Other engine types may appear in only one or some of the engine clusters **214**, as needed to satisfy their performance requirements or the performance requirements of the pipeline processor system **20**.

[0752] FIG. **38** illustrates a genomics pipeline processor system **30**, showing, in addition to the engine clusters described above, one or more embedded central processing units (CPUs) **302**. Examples of such embedded CPUs include Snapdragons® or standard ARM® cores. These CPUs execute fully programmable bio-IT algorithms, such as advanced variant calling. Such processing is accelerated by computing functions in the engine clusters, which can be called by the CPU cores **302** as needed. Furthermore, even engine-centric processing, such as mapping and alignment, can be managed by the CPU cores **302**, giving them heightened programmability.

[0753] FIG. **39** illustrates a processing flow for a genomics pipeline processor system and method. In some preferred implementations, there are three passes over the data. The first pass includes mapping **402** and alignment **404**, with the full set of reads streamed through the engines. The second pass includes sorting **406**, where one large block to be sorted (e.g., a substantial portion or all reads previously mapped to a single chromosome) is loaded into memory, sorted by the processing engines, and returned to the host. The third pass includes downstream stages (local realignment **408**, duplicate marking **410**, base quality score recalibration (BQSR) **412**, BAM output **414**, reduced BAM output **416**, and/or CRAM compression **418**). The steps and functions of the third pass may be done in any combination or subcombination, and in any order, in a single pass. A virtual pipeline architecture, such as described above, is used to stream reads from the host into circular buffers in memory, through one processing engine after another in sequence, and back out to the host. In some implementations, CRAM decompression can be a separate streaming function. In some implementations, the BAM output **414**, reduced BAM output **416**, and/or CRAM compression **418** can be replaced with variant calling, compression and decompression.

[0754] In various instances, a hardware implementation of a sequence analysis pipeline is described. This can be done in a number of different ways such as an FPGA or ASIC or structured ASIC implementation. The functional blocks that are implemented by the FPGA or ASIC or structured ASIC are set forth in FIG. **40**. Accordingly, the system includes a number of blocks or modules to do sequence analysis. The input to the hardware realization can be a FASTQ file, but is not limited to this format. In addition to the FASTQ file, the input to the FPGA or ASIC or structured ASIC consists of side information, such as Flow Space Information from technology such as the Ion Torrent. The blocks or modules may include the following blocks: Error Control, Mapping, Alignment, Sorting, Local Realignment, Duplicate Marking, Base Quality Recalibration, BAM and Side Information reduction and/or variant calling.

[0755] These blocks or modules can be present inside, or implemented by, the hardware, but some of these blocks may be omitted or other blocks added to achieve the purpose of realizing a sequence analysis pipeline. Blocks **2** and **3** describe two alternatives of the sequence analysis pipeline platform. The sequence analysis pipeline platform comprising an FPGA or ASIC or structured ASIC and software assisted by a host (e.g., PC, server, cluster or cloud computing) with cloud and/or cluster storage. Blocks **4-7** describe different interfaces that the sequence analysis pipeline can have. In Blocks **4** and **6** the interface can be a PCIe interface, but is not limited to a PCIe interface. In Blocks **5** and **7** the hardware (FPGA or ASIC or structured ASIC) can be directly integrated into a sequencing machine. Blocks **8** and **9** describe the integration of the hardware sequence analysis pipeline integrated into a host system such as a PC, server cluster or sequencer. Surrounding the hardware FPGA or ASIC or structured ASIC are lots of DDR3 memory elements and a PCIe interface. The board with the FPGA/ASIC/sASIC connects to a host computer, consisting of a host CPU, that could be either a low power CPU such as an ARM®, Snapdragon®, or any other processor. Block **10** illustrates a hardware sequence analysis pipeline API that can be accessed by third party applications to perform tertiary analysis.

[0756] Accordingly, as depicted in FIG. **41**, in various embodiments, an apparatus of the disclosure may include a computing architecture, such as embedded in a silicon field gated programmable array (FPGA) or application specific integrated circuit (ASIC) **100**. The FPGA **100** can be integrated into a printed circuit board (PCB) **104**, such as a Peripheral Component Interface—Express (PCIe) card, that can be plugged into a computing platform. In various instances, as shown in FIG. **43**, the PCIe card **104** may include a single FPGA **100**, which FPGA may be surrounded by local memories **105**, however, in various embodiments, the PCIe card **104** may include a plurality of FPGAs **100**A, **100**B and **100**C. In various instances, the PCI card may also include a PCIe bus. This PCIe card **104** can be added to a computing platform to execute algorithms on extremely large data sets. Accordingly, in various instances, the overall work flow of genomic sequencing involving the FPGA may include the following: Sample preparation, Alignment (including mapping and alignment), Variant analysis, Biological Interpretation, and/or Specific Applications.

[0757] Hence, in various embodiments, an apparatus of the disclosure may include a computing architecture that achieves the high performance execution of algorithms, such as mapping and alignment algorithms, that operate on extremely large data sets, such as where the data sets exhibit poor locality of reference (LOR). These algorithms are designed to reconstruct a whole genome from millions of short read sequences, from modern so-called next generation sequencers, require multi-gigabyte data structures that are randomly accessed. Once reconstruction is achieved, as described herein above, further algorithms with similar characteristics are used to compare one genome to libraries of others, do gene function analysis, etc.

[0758] Currently, there are two major approaches that in general may be implemented, such as purpose multicore CPUs and general purpose Graphic Processing Units (GPGPUs). In such an instance ach CPU in a multicore system may have a classical cache based architecture, wherein instructions and data are fetched from a level 1 cache (L1 cache) that is small but has extremely fast access. Multiple L1 caches may be connected to a larger but slower shared L2 cache. The L2 cache may be connected to a large but slower DRAM (Dynamic Random Access Memory) system memory, or may be connected to an even larger but slower L3 cache which may then connected to DRAM. An advantage of this arrangement may be that applications in which programs and data exhibit locality of reference behave nearly as if they are executing on a computer with a single memory as large as the DRAM but as fast as the L1 cache. Because full custom, highly optimized CPUs operate at very high clock rates, e.g., 2 to 4 GHz, this architecture may be essential to achieving good performance.

[0759] Further, GPGPUs may be employed to extend this architecture, such as by implementing very large numbers of small CPUs, each with their own small L1 cache, wherein each CPU executes the same instructions on different subsets of the data. This is a so called SIMD (Single Instruction stream, Multiple Data stream) architecture. Economy is gained by sharing the instruction fetch and decode logic across a large number of CPUs. Each cache has access to multiple large external DRAMs via an interconnection network. Assuming the computation to be performed is highly parallelizable, GPGPUs have a significant advantage over general purpose CPUs due to having large numbers of computing resources. Nevertheless, they still have a caching architecture and their performance is hurt by applications that do not have a high enough degree of locality of reference. That leads to a high cache miss rate and processors that are idle while waiting for data to arrive from the external DRAM.

[0760] For instance, in various instances, Dynamic RAMs may be used for system memory because they are more economical than Static RAMs (SRAM). The rule of thumb used to be that DRAMs had 4× the capacity for the same cost as SRAMs. However, due to declining demand for SRAMs in favor of DRAMs, which difference has increased considerably due to the economies of scale that favor DRAMs which are in high demand. Independent of cost, DRAMs are 4× as dense as SRAMs laid out in the same silicon area because they only require one transistor and capacitor per bit compared to 4 transistors per bit to implement the SRAM's flip-flop. The DRAM represents a single bit of information as the presence or absence of charge on a capacitor. A problem with this arrangement is that the charge decays over time, so it has to be refreshed periodically. The need to do this has led to architectures that organize the memory into independent blocks and access mechanisms that deliver multiple words of memory per request. This compensates for times when a given block is unavailable while being refreshed. The idea is to move a lot of data while a given block is available. This is in contrast to SRAMs in which any location in memory is available in a single access in a constant amount of time. This characteristic allows memory accesses to be single word oriented rather than block oriented. DRAMs work well in a caching architecture because each cache miss leads to a block of memory being read in from the DRAM. The theory of locality of reference is that if just accessed word N, then probably going to access words N+1, N+2, N+3 and so on, soon.

[0761] FIG. **42** presents an alternative embodiment to FIG. **41**, having a multiplicity of chips **100**A, **100**B, **100**C, where each chip may include one or more of the various genomics and/or bioinformatics processing modules, e.g., of an exemplary pipeline analysis apparatus, as disclosed herein.

[0762] FIG. **43** illustrates a system **500** for executing a sequence analysis pipeline on genetic sequence data. The system **500** includes a configuration manager **502** that includes a computing system. The computing system of the configuration manager **502** can include a personal computer or other computer workstation, or can be implemented by a suite of networked computers. The configuration manager **502** can further include one or more third party applications connected with the computing system by one or more APIs, which, with one or more proprietary applications, generate a configuration for processing genomics data from a sequencer or other genomics data source. The configuration manager **502** further includes drivers that load the configuration to the genomics pipeline processor system **10**. The genomics pipeline processor system **10** can output result data to, or be accessed via, the Web **504** or other network, for storage of the result data in an electronic health record **506** or other knowledge database **508**.

[0763] As discussed in several paces herein above, the chip implementing the genomics pipeline processor can be connected or integrated in a sequencer. The chip can also be connected or integrated on an expansion card, e.g. PCIe, and the expansion card can by connected or integrated in a sequencer. In other implementations, the chip can be connected or integrated in a server computer that is connected to a sequencer, to transfer genomic reads from the sequencer to the server. In yet other implementations, the chip can be connected or integrated in a server in a cloud computing cluster of computers and servers. A system can include one or more sequencers connected (e.g. via Ethernet) to a server containing the chip, where genomic reads are generated by the multiple sequencers, transmitted to the server, and then mapped and aligned in the chip.

[0764] For instance, in general next generation DNA sequencer (NGS) data pipelines, the primary analysis stage processing is generally specific to a given sequencing technology. This primary analysis stage functions to translate physical signals detected inside the sequencer into "reads" of nucleotide sequences with associated quality (confidence) scores, e.g. FASTQ format files, or other formats containing sequence and usually quality information. After such a format is achieved, secondary analysis proceeds, as described herein, to determine the content of the sequenced sample DNA (or RNA etc.), such as by mapping and aligning reads to a reference genome, sorting, duplicate marking, base quality score recalibration, local re-alignment, and variant calling. Tertiary analysis may then follow, to extract medical or research implications from the determined DNA content.

[0765] However, primary analysis, as mentioned above, is often quite specific in nature to the sequencing technology employed. In various sequencers, nucleotides are detected by sensing electrical charges, electrical currents, or radiated light. Some primary analysis pipelines often include: Signal processing to amplify, filter, separate, and measure sensor output; Data reduction, such as by quantization, decimation, averaging, transformation, etc.; Image processing or numerical processing to identify and enhance meaningful signals, and associate them with specific reads and nucleotides (e.g. image offset calculation, cluster identification); Algorithmic processing and heuristics to compensate for sequencing technology artifacts (e.g. phasing estimates, cross-talk matrices); Bayesian probability calculations; Hidden Markov models; Base calling (selecting the most likely nucleotide at each position in the sequence); Base call quality (confidence) estimation, and the like.

[0766] Primary analysis can be extremely commutatively intensive, sometimes as intensive as secondary analysis. For instance, in existing sequencing technologies, primary analysis often utilizes FPGAs and/or GPUs to accelerate processing beyond CPU capabilities. But these accelerated functions can be performed much more efficiently in custom integrated circuitry, such as that described herein. For example, they can be implemented in a structured ASIC using the configurable metal layers, as they do not require as much physical layout precision as embedded processor cores; however, the massively parallel computation implemented in large FPGAs and GPUs may be difficult to fit in the configurable structured ASIC resources. An alternative is to implement primary processing acceleration logic in the master slice of a structured ASIC, taking advantage of the standard cell space efficiency in the master slice.

[0767] A reason that secondary processing functions may be implemented in a structured ASIC configurable metal layers is that secondary genomic data processing algorithms are still evolving via active research. It may be therefore beneficial to be able to inexpensively produce a freshly updated structured ASIC design periodically, such as every year or every two years, to utilize the latest algorithms. By contrast, primary analysis algorithms currently employed are more mature, the necessary processing having been researched and defined by the respective sequencer manufacturers. Even to the extent it is still subject to change, the algorithms are more generic signal and numerical processing than is the case in secondary analysis, so that appropriate configurability and micro-coding of primary processing acceleration modules can make them flexible enough to accommodate significant changes. If present, embedded processor cores increase this flexibility even further. For these reasons, it is reasonable to design primary processing acceleration modalities into a structured ASIC master slice, as herein described.

[0768] It is also advantageous to integrate primary processing acceleration and secondary processing acceleration in a single integrated circuit (standard cell or structured ASIC), with or without embedded processors. This may be beneficial because sequencers produce data requiring both primary and secondary analysis, and integrating them in a single device is most efficient in terms of cost, space, power, and resource sharing. If embedded processors are also present, they can be leveraged to increase the speed and flexibility of both primary and secondary processing.

[0769] These three components—primary accelerators, secondary accelerators, and embedded processors—can be implemented in a structured ASIC master slice, and/or using configurable metal layers, in any combination. All three could be in the master slice, or all three could use configurable metal layers, or any one or two of them could be in the master slice, and the others use configurable metal layers. In any of these configurations, all three can communicate with each other, in any combination, directly and/or via memory, and cooperate in common tasks. One advantageous configuration is to implement primary acceleration and embedded processors in the master slice, and implement secondary acceleration using configurable metal layers.

[0770] Additionally, as indicated above, the chip, whether implemented as an ASIC, FPGA, or a structured ASIC, may include or otherwise be associated with one or more memory architectures. For instance, a memory architecture can consist of M memory modules that interface with the chip, such as with an ASIC. The ASIC may be implemented using many different technologies, including FPGAs (Field Programmable Gate Arrays) or structured ASIC, standard cells, or full custom logic. Within the ASIC are a Memory Subsystem (MSS) and Functional Processing Units (FPUs). The MSS contains M memory controllers (MCs) for the memory modules, N system memory interfaces (SMIs) for the FPUs, and an N×M crossbar that allows any SMI to access any MC. Arbitration is provided in the case of contention.

[0771] Each memory module is constructed from DRAM chips that are addressed by an A_{MM} bit word and support data transfers D_{MM} bits wide. The memory has $2^{A_{MM}}$ address locations. A key characteristic of DRAM is that it performs reads/writes in W word bursts using the supplied address as the base address, B, and fetching or storing locations B+1, B+2, . . . B+W−1 as well. A typical value for W is 8.

[0772] In the MSS of the ASIC, each memory controller supplies the required control signals and performs any necessary multiplexing/demultiplexing between the system word width, D_{SYS}, and the memory word width, D_{MM}, as well as handling the requirements for read/write bursts. It can contain extra buffering so that multiple memory requests can be queued up and processed in a pipelined fashion to maximize throughput. This compensates for multiple clock cycles of latency between presentation of an address and completion of a memory operation (read or write).

[0773] The MC may operate at the speed of the attached DRAM in a memory module. Assume its clock rate is C_{MM}. This is often several times faster than the core speed at which the majority of the logic in the ASIC operates which is C_{SYS}. Hence the multiplexing/demultiplexing logic is placed close to its associated interface pins to minimize signal distances. Demultiplexing is the first operation performed on incoming data and multiplexing is the last operation performed on outgoing data. The remainder of the MSS operates on D_{SYS} width data which is wider than D_{MM}, enabling use of the slower C_{SYS} clock speed.

[0774] Each system memory interface in the MSS presents an A_{SYS} bit address bus and a D_{SYS} bit data bus to any attached FPU. The SMI is designed to make it appear to an attached FPU that it has random access to a single large fast memory. The FPU has no awareness of the existence of separate memory modules. A_{SYS} is large enough to allow access to any memory location in any attached memory module. The mapping from system address space to memory module address space is explained below.

[0775] The N system memory interfaces are cross connected to the M memory modules via an N×M crossbar. The crossbar provides min(M,N) simultaneous connections among the SMIs and MCs, provides arbitration for conflicts, and facilitates translation of system address space into memory module address space.

[0776] The organization of FPUs is highly flexible. One or more FPUs can share the same system memory interface. To maximize performance, FPUs that do not operate at the same time should share an SMI. Those that operate concurrently, should be attached to different SMIs. An FPU that operates on a data structure larger than D_{SYS} can use multiple SMIs to access the whole data structure in a single memory operation. Hence this memory architecture supports a wide range of computation architectures. Each FPU may be identical and thus an array of them may be implemented in a two dimensional structure. This is illustrated where FPU (i,j) is the j^{th} unit attached to SMI i, $0 \leq i < N$, $0 \leq j < k_i$. In this case, all the k_i are the same size and k_i may be as small as 1. This supports SIMD (single instruction stream, multiple data stream) and MIMD architectures (multiple instruction stream, multiple data stream) depending on whether the FPUs fetch instructions from the same or individual instruction memories.

[0777] In one particular aspect, the disclosure is directed to a system, such as to a system for executing a sequence analysis pipeline on genetic sequence data. In various instances, the system may include an electronic data source, such as a data source that provides digital signals, for instance, digital signals representing a plurality of reads of genomic data, where each of the plurality of reads of genomic data include a sequence of nucleotides. The system may include one or more of a memory, such as a memory storing one or more genetic reference sequences and/or an index of the one or more genetic reference sequences; and/or the system may include a chip, such as an ASIC, FPGA, or sASIC.

[0778] More particularly, in various particular embodiments, the system may include a structured application specific integrated circuit (ASIC), such as where the chip is formed of a set of mask-programmable, hardwired digital logic circuits that may be interconnected by a plurality of physical electrical interconnects. In various instances, one or more of the plurality of physical electrical interconnects include an input to the structured ASIC that is connected with the electronic data source, such as for receiving the plurality of reads of genomic data. In such an instance, one or more of the plurality of physical electrical interconnects may include a memory interface for the structured ASIC to access the memory. Accordingly, the hardwired digital logic circuits may be arranged as a set of processing engines, such as where each processing engine may be formed of a subset of the hardwired digital logic circuits so as to perform one or more steps in the sequence analysis pipeline on the plurality of reads of genomic data. In various embodiments, one or more, e.g., each, subset of the hardwired digital logic circuits may be in a wired configuration such as to perform the one or more steps in the sequence analysis pipeline. For instance, the set of processing engines may be configured to include one or more of a mapping module, an alignment module, and/or a sorting module.

[0779] For example, the set of processing engines may include a mapping module that is in the wired configuration, and is configured to access, according to at least some of the sequence of nucleotides in a read of the plurality of reads, the index of the one or more genetic reference sequences from the memory via the memory interface so as to map the read to one or more segments of the one or more genetic reference sequences based on the index. For instance, in certain embodiments, the index of the one or more genetic reference sequences may include a hash table, and/or the mapping module may apply a hash function to the at least some of the sequence of nucleotides to access the hash table of the index.

[0780] The processing engines may also or alternatively include an alignment module that is in the wired configuration, and is configured to access the one or more genetic reference sequences from the memory, e.g., via the memory interface, so as to align the read to one or more positions in the one or more segments of the one or more genetic reference sequences, such as obtained from the mapping module. The processing engines may also or alternatively include a sorting module that is in the wired configuration, and is configured to access the one or more aligned reads from the memory, e.g., via the memory interface, so as to sort the read to one or more positions, e.g., chromosomal positions, in the genetic reference sequences, such as obtained from the alignment module.

[0781] In various instances, the structured ASIC may include a master slice that incorporates at least some of the hardwired digital logic circuits, and in some instances, may include one or more configurable metal layers that are formed on the master slice, such as where each of the one or more configurable metal layers may have at least some of the plurality of physical electrical interconnects that interconnect the at least some of the hardwired digital logic circuits to form the set of processing engines. In certain embodiments, one or more of the plurality of physical electrical interconnects may include an output from the structured ASIC, such as for communicating result data from the mapping module and/or the alignment module and/or sorting module. In certain instance,

[0782] In various instances, the structured ASIC may include a master controller to establish the wired configuration for each subset of the hardwired digital logic circuits so as to perform the one or more steps in the sequence analysis pipeline. In various embodiments, the wired configuration is established upon manufacture of the integrated circuit and is non-volatile. In some embodiments, the structured ASIC and/or the memory are housed on an expansion card, such as a peripheral component interconnect (PCI) card. As indicated above, in various embodiments, the system may include a sequencer, such as where the sequencer includes the electronic data source that provides the digital signals representing the plurality of reads of genomic data. And in such an instance, the expansion card may be physically integrated with the sequencer.

[0783] Additionally, in various embodiments, a structured application-specific integrated circuit (ASIC) may be provided, such as for analyzing genetic sequence data, such as where the genetic sequence data is stored in a memory, such as a memory storing one or more genetic reference sequences associated with genomic data, and/or an index of the one or more genetic reference sequences. In such an instance, the structured ASIC may include a master slice that further includes a set of digital logic circuits, and may additionally include one or more configurable metal layers that are formed on the master slice, such as where each of the one or more configurable metal layers may have a set of wired connections arranged to interconnect a subset of the digital logic circuits to form a set of processing engines. In such an instance, the set of processing engines may include a mapping engine, an alignment engine, and/or a sorting engine. In various instances, a portion of the set of digital logic circuits in the master slice is hardwired as a base calling engine.

[0784] For instance, the set of processing engines may include a mapping engine to access, e.g., according to at least some of the sequence of nucleotides in a read of the plurality of reads, the index of the one or more genetic reference sequences stored in the memory, so as to map the read to one or more segments of the one or more genetic reference sequences, e.g., based on the index. Additionally or alternatively, the set of processing engines may include an alignment engine such as to access the one or more genetic reference sequences from the memory, e.g., via the memory interface, so as to align the read to one or more positions in the one or more segments of the one or more genetic reference sequences from the mapping module. Additionally, or alternatively the set of processing engines may include a sorting engine to sort each aligned read according to the one or more positions in the one or more genetic reference sequences.

[0785] In one embodiment, a system for executing a sequence analysis pipeline on genetic sequence data is provided where the system includes an electronic data source that provides digital signals representing a plurality of reads of genomic data, such as where each of the plurality of reads of genomic data include a sequence of nucleotides. The system may include one or more of a memory, e.g., for storing one or more genetic reference sequences and/or an index of the one or more genetic reference sequences; and/or the system may include an integrated circuit having a master slice, such as a master slice formed by a photolithographic mask that defines a set of digital logic circuits. In such an instance, the master slice may be configured for having one or more functions, as those described herein above, integrated therein. For instance, the master slice may have one or more configurable metal layers, such as where each of the one or more configurable metal layers has one or more conductive interconnects that connect a subset of the set of digital logic circuits in a wired configuration to perform the aforesaid functions.

[0786] In various aspects, a method of making a structured application-specific integrated circuit (ASIC) for analyzing genetic sequence data is provided. In certain embodiments, the method includes one or more of providing a plurality of photolithographic masks, such as masks that define a set of digital logic circuits of a master slice; forming the set of digital logic, such as by using the plurality of photolithographic masks to form the master slice; providing two or more different sets of design-specific configurable metal layer masks, such as masks that define corresponding two or more digital logic to implement a set of processing engines; forming two or more configurable metal layers, such as using two or more different sets of design-specific configurable metal layer masks, for instance, where each of the two more configurable metal layers have a set of wired connections that may be arranged according to a design of the configurable metal layer masks, for example, to interconnect a subset of the digital logic circuits to form a set of processing engines; and/or providing the two or more configurable metal layers onto the master slice to form the set of processing engines.

[0787] One or more aspects or features of the subject matter described herein can be realized in digital electronic circuitry, integrated circuitry, specially designed application specific integrated circuits (ASICs), field programmable gate arrays (FPGAs), or structured ASIC computer hardware, firmware, software, and/or combinations thereof.

[0788] These various aspects or features can include implementation in one or more computer programs that are executable and/or interpretable on a programmable system including at least one programmable processor, which can be special or general purpose, coupled to receive data and instructions from, and to transmit data and instructions to, a storage system, at least one input device, and at least one output device. The programmable system or computing system may include clients and servers. A client and server are generally remote from each other and typically interact through a communication network. The relationship of client and server arises by virtue of computer programs running on the respective computers and having a client-server relationship to each other.

[0789] These computer programs, which can also be referred to as programs, software, software applications, applications, components, or code, include machine instructions for a programmable processor, and can be implemented in a high-level procedural and/or object-oriented programming language, and/or in assembly/machine language. As used herein, the term "machine-readable medium" refers to any computer program product, apparatus and/or device, such as for example magnetic discs, optical disks, memory, and Programmable Logic Devices (PLDs), used to provide machine instructions and/or data to a programmable processor, including a machine-readable medium that receives machine instructions as a machine-readable signal. The term "machine-readable signal" refers to any signal used to provide machine instructions and/or data to a programmable processor. The machine-readable medium can store such machine instructions non-transitorily, such as for example as would a non-transient solid-state memory or a magnetic hard drive or any equivalent storage medium. The machine-readable medium can alternatively or additionally store such machine instructions in a transient manner, such as for example as would a processor cache or other random access memory associated with one or more physical processor cores.

[0790] To provide for interaction with a user, one or more aspects or features of the subject matter described herein can be implemented on a computer having a display device, such as for example a cathode ray tube (CRT), a liquid crystal display (LCD) or a light emitting diode (LED) monitor for displaying information to the user and a keyboard and a pointing device, such as for example a mouse or a trackball, by which the user may provide input to the computer. Other kinds of devices can be used to provide for interaction with a user as well. For example, feedback provided to the user can be any form of sensory feedback, such as for example visual feedback, auditory feedback, or tactile feedback; and input from the user may be received in any form, including, but not limited to, acoustic, speech, or tactile input. Other possible input devices include, but are not limited to, touch screens or other touch-sensitive devices such as single or multi-point resistive or capacitive trackpads, voice recognition hardware and software, optical scanners, optical pointers, digital image capture devices and associated interpretation software, and the like.

[0791] The subject matter described herein can be embodied in systems, apparatus, methods, and/or articles depending on the desired configuration. The implementations set forth in the foregoing description do not represent all implementations consistent with the subject matter described herein. Instead, they are merely some examples consistent with aspects related to the described subject matter. Although a few variations have been described in detail above, other modifications or additions are possible. In particular, further features and/or variations can be provided in addition to those set forth herein. For example, the implementations described above can be directed to various combinations and subcombinations of the disclosed features and/or combinations and subcombinations of several further features disclosed above. In addition, the logic flows depicted in the accompanying figures and/or described herein do not necessarily require the particular order shown, or sequential order, to achieve desirable results. Other implementations may be within the scope of the following claims.

What is claimed is:

1. A system for executing a Hidden Markov Model (HMM) analysis on genetic sequence data, the genetic sequence data including a read of genomic data and a reference and/or haplotype sequence, the system comprising:

one or more memories for storing the read of genomic sequence data and the reference and/or haplotype sequence, each of the read of genomic sequence data and the reference and/or haplotype sequence comprising a sequence of nucleotides; and

an integrated circuit formed of a plurality of hardwired digital logic circuits that are interconnectable by a plurality of physical electrical interconnects, one or more of the plurality of physical electrical interconnects comprising a memory interface for the integrated circuit to access the memory, the plurality of hardwired digital logic circuits including at least a first subset of hardwired digital logic circuits, the first subset of hardwired digital logic circuits being arranged as a first set of processing engines, a first set of processing engines to perform one or more steps in the HMM analysis on the read of genomic sequence data and the haplotype sequence, the first set of processing engines comprising an HMM module in a first configuration of the subset of hardwired digital logic circuits to access from the one or more memories, via the memory interface, at least some of the sequence of nucleotides in the read of genomic sequence data and the haplotype sequence data, and to perform the HMM analysis on the at least some of the sequence of nucleotides in the read of genomic sequence data and the at least some of the sequence of nucleotides in the haplotype sequence to produce HMM result data; and

one or more of the plurality of physical electrical interconnects comprising an output from the integrated circuit for communicating the HMM result data from the HMM module.

* * * * *