(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
29 June 2017 (29.06.2017)

WIPO | PCT

(10) International Publication Number
WO 2017/112226 A1

(51) International Patent Classification:
G06F 9/50 (2006.01)

(21) International Application Number:
PCT/US2016/063170

(22) International Filing Date:
21 November 2016 (21.11.2016)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
14/976,360 21 December 2015 (21.12.2015) US

(71) Applicant: **INTEL CORPORATION** [US/US]; 2200 Mission College Boulevard, Santa Clara, California 95054 (US).

(72) Inventors: **THIERRY, Philippe**; Les Montalets, 2 Rue de Paris, 92190 Meudon (FR). **OULD-AHMED-VALL, Elmoustapha**; 5000 W Chandler Blvd, M/S: CH7-401, Chandler, Arizona 85226 (US).

(74) Agents: **PORTNOVA, Marina** et al.; Lowenstein Sandler LLP, 65 Livingston Avenue, Roseland, New Jersey 07068 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JP, KE, KG, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU,

[Continued on next page]

(54) Title: SCHEDULING HIGHLY PARALLEL APPLICATIONS

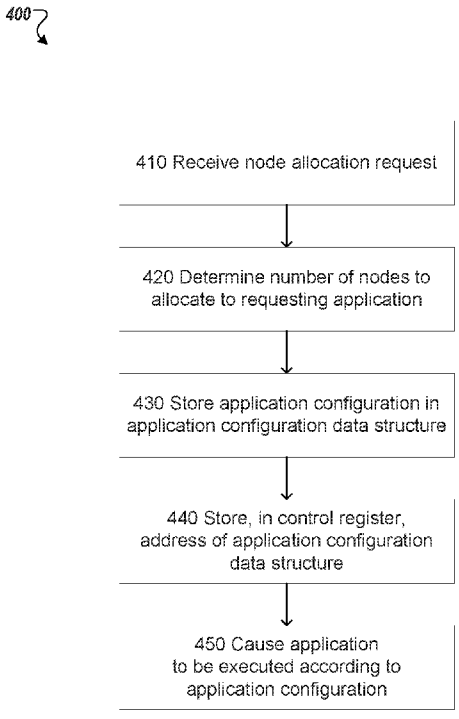


FIGURE 4

(57) Abstract: Systems and methods for scheduling highly-parallel applications executed by highperformance computing systems. An example processing system may comprise: a control register and a processing core, communicatively coupled to the control register. The processing core may be configured to receive a node allocation request specifying an expected running time of an application and a requested number of nodes of a cluster of nodes; determine, in view of the node allocation request and a current load on the plurality of nodes, an actual number of nodes to be allocated to the application, wherein the actual number of nodes to be allocated to the application optimizes a cluster load criterion; and notify, using the control register, the application of the actual number of nodes.

WO 2017/112226 A1

LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

— *with international search report (Art. 21(3))*

SCHEDULING HIGHLY PARALLEL APPLICATIONS

[0001] The present disclosure generally relates to high-performance computer (HPC) systems and, more specifically, to instruction set architecture support for scheduling highly parallel applications.

Background

[0002] A high-performance computing (HPC) system may include multiple compute nodes grouped into one or more compute clusters. The compute nodes may be managed by a control node.

Brief Description of the Drawings

[0003] The present disclosure is illustrated by way of examples, and not by way of limitation, and may be more fully understood with references to the following detailed description when considered in connection with the figures.

[0004] Fig. 1 schematically illustrates a block diagram of an example high-performance computing system operating in accordance with one or more aspects of the present disclosure.

[0005] Figs. 2A-2B schematically illustrate various cluster configurations, in accordance with one or more aspects of the present disclosure.

[0006] Fig. 3 is a block diagram of an example processing system operating in accordance with one or more aspects of the present disclosure.

[0007] Fig. 4 depicts a flowchart of an example method 400 for determining the application configuration in a cluster of nodes, in accordance with one or more aspects of the present disclosure.

[0008] Fig. 5 illustrates packed data types, in accordance with one or more aspects of the present disclosure.

[0009] Fig. 6A illustrates elements of a processor micro-architecture, in accordance with one or more aspects of the present disclosure.

[0010] Fig. 6B illustrates elements of a processor micro-architecture, in accordance with one or more aspects of the present disclosure.

[0011] Fig. 7 is a block diagram of a processor operating in accordance with one or more aspects of the present disclosure.

[0012] Fig. 8 is a block diagram of a processor operating in accordance with one or more aspects of the present disclosure.

[0013] Fig. 9 is a block diagram of a system-on-a-chip operating in accordance with one or more aspects of the present disclosure.

[0014] Fig. 10 is a block diagram of a computer system operating in accordance with one or more aspects of the present disclosure.

[0015] Fig. 11 illustrates another example block diagram for a computing system operating in accordance with one or more aspects of the present disclosure.

[0016] Fig. 12 illustrates a diagrammatic representation of a machine in the example form of a computing system operating in accordance with one or more aspects of the present disclosure.

Detailed Description

[0017] Described herein are systems and methods for scheduling highly parallel applications.

[0018] A high-performance computing (HPC) system may include multiple compute nodes grouped into one or more compute clusters. A batch scheduler residing on a dedicated node or on one of the compute nodes may be employed to schedule computing jobs (also referred to as applications) for execution on the compute nodes. One or more compute nodes may be requested for executing a given application, and an expected run time may be specified. The scheduler may utilize these and other execution parameters to assign the applications to compute nodes for execution.

[0019] The scheduler may employ scheduling strategies that maximize the usage level of the compute nodes, in order to increase the overall system cost efficiency. However, in common implementations, the average usage of compute nodes rarely exceeds 80% of their capacity, since the number of nodes that is available for scheduling at a given moment in time may not necessary be equal to a sum of requested number of nodes of one or more applications to be scheduled. In other words, the difference between the available number of nodes and the number of nodes allocated to one or more applications may be less than the number of nodes that would be required to schedule another application that is waiting for execution.

[0020] Under-utilization of compute nodes adversely affects the overall system cost-efficiency, since even an inactive compute node consumes electric energy for its operation and cooling. Utilizing at 80% capacity a cluster of compute nodes that consumes between 20 and 40 megawatt (MW) of electric energy means that 4-8 MW of electric energy would be lost every hour.

[0021] The present disclosure addresses the above-noted and other deficiencies by providing instruction set architecture (ISA) support for a batch scheduling method that causes the applications to increase or decrease the application footprint (i.e., the number of compute nodes) dynamically at runtime. In accordance with one or more aspects of the present disclosure, a processor executing the batch scheduler may implement a control register for storing an address

of a memory data structure reflecting the current application configuration (e.g., identifiers of the nodes that are assigned to the application). The processor may further implement several instructions to manipulate the control register. Such instructions may include Load Configuration (LDCONFIG) for reading the configuration, Store Configuration (STCONFIG) for storing the configuration, and Request Configuration Ownership (REQCONFIG) for requesting a lock of the application configuration data structure.

[0022] A batch scheduler operating in accordance one or more aspects of the present disclosure may be programmed to determine cluster configurations that optimize a certain cluster load criterion. In an illustrative example, the batch scheduler may receive a node allocation request from an application that is waiting to be executed or is being executed by the high-performance computing system. The node allocation request may specify the expected application and the requested number of nodes for executing the application.

[0023] Responsive to receiving the node allocation request, the batch scheduler may determine the cluster configuration that optimizes a chosen cluster load criterion. The batch scheduler may then utilize the processor control register to notify the application of the actual number of nodes that have been assigned to the application, as described in more details herein below.

[0024] Various aspects of the above referenced methods and systems are described in details herein below by way of examples, rather than by way of limitation.

[0025] Referring now to Fig. 1, example high-performance computing system 1000 may include a master node 110 and a plurality of compute nodes 120A-120N interconnected by a network 135. Master node 110 may execute a batch scheduler 140, which may communicate to each of the nodes 120A-120N, e.g., via Message Passing Interface (MPI) messages 150.

[0026] Master node 110 may implement an input job queue 160 for queuing applications 170A-170K waiting to be scheduled. Each application may be described by a respective application identifier, the number of compute node requested for executing the application, and the expected runtime of the application.

[0027] Batch scheduler 140 may be programmed to determine cluster configurations that optimize a certain cluster load criterion. As schematically illustrated by Fig. 2A, batch scheduler 140 may generate a plurality of combinations of applications 170A-170K, and select a combination that minimizes the number of nodes that remain unassigned to any application at any given time. Alternatively, batch scheduler 140 may select a combination that minimizes the total processing capacity that remains unused within a given period of time. Fig. 2 schematically illustrates an example of scheduling a plurality of applications 170A-170K to be executed by a plurality of compute nodes 120A-120N within a certain time period. As shown in Fig. 2A,

conventional scheduling algorithms may produce schedules that have significant levels of unused processing capacities (denoted as blocks 180).

[0028] Since some applications (e.g., client-server applications) may have an architecture that is easily adaptable to the dynamically changing number of nodes, the batch scheduler 140 may improve the cluster efficiency by employing scheduling methods that causes the applications to increase or decrease the application footprint (i.e., the number of compute nodes) dynamically at runtime. Such scheduling methods may produce more efficient, as compared with conventional methods, schedules that further minimize the levels of unused processing capacities, as schematically illustrated by Fig. 2B.

[0029] Fig. 3 schematically illustrates a block diagram of an example processing system operating in accordance with one or more aspects of the present disclosure. As schematically illustrated by Fig. 3, processing system 100 may include one or more processors 102 providing ISA support for a batch scheduling method that instructs the applications to increase or decrease the application footprint dynamically at runtime, in accordance with one or more aspects of the present disclosure.

[0030] In the illustrative example of Fig. 3, processor 102 includes one or more execution units 408 configured to perform instructions of a certain instruction set. Processor 102 is coupled to a processor bus 410 that transmits data signals between processor 102 and other components of system 100. Other elements of system 100 (e.g. graphics video card 412, memory controller hub 416, memory 420, I/O controller hub 430, wireless transceiver 426, flash BIOS 428, network controller 434, audio controller 436, serial expansion port 438, legacy I/O controller 440, etc.) may perform their conventional functions that are well known to those familiar with the art.

[0031] Each processor 102 may comprise a plurality of registers, including general purpose registers and specialized registers. In certain implementations, processing system 100 may also include various other components not shown in Fig. 3. More detailed description of various components of processing system 100 is presented herein below.

[0032] Processor 102 may comprise a control register 122 that may be utilized for storing an address of a memory data structure reflecting the current application configuration. In an illustrative example, the application configuration data structure may store an application identifier and the number of nodes to be assigned for executing the application. In an illustrative example, the application configuration data structure may initially be created by the application, which may record the requested number of nodes. The batch scheduler may eventually modify the number of nodes assigned to the application, in an attempt to optimize a cluster load criterion.

[0033] Alternatively, the application configuration data structure may be created by the batch scheduler, which may record the number of nodes assigned to the application. The creator of the application configuration data structure may store the address of the data structure in control register 122 using STCONFIG instruction. The application may read the content of control register 122 using LDCONFIG instruction, and may retrieve the address of the data structure storing the current configuration of nodes for the application.

[0034] In certain implementations, processor 102 may further implement Request Configuration Ownership (REQCONFIG) for requesting a lock of the application configuration data structure before modifying the data structure. In an illustrative example, the lock may be released upon expiration of a pre-defined timeout since obtaining the lock. Alternatively, the lock may be released explicitly, by executing REQCONFIG command with a parameter indicating the lock release.

[0035] In certain implementations, the batch scheduler may determine that an application failed to release, within a pre-determined period of time, a previously obtained lock of the application configuration data structure. The batch scheduler may deem the application to be non-responsive and may initiate an appropriate recovery procedure (e.g., by terminating the non-responsive application).

[0036] As noted herein above, the batch scheduler operating in accordance one or more aspects of the present disclosure may be programmed to determine cluster configurations that optimize a certain cluster load criterion. In an illustrative example, the batch scheduler may receive a node allocation request from an application that is waiting to be executed or is being executed by the high-performance computing system. The node allocation request may specify the expected application and the requested number of nodes for executing the application. In certain implementations, the node allocation request may further specify the minimum number of nodes required, a waiting preference parameter indicating whether an additional waiting time is preferred over allocation of a lesser number of nodes than requested, application priority, and/or various other application parameters.

[0037] Responsive to receiving the node allocation request, the batch scheduler may determine the actual number of nodes to be allocated to the application. As noted herein above, the actual number of nodes may be chosen to optimize a certain cluster load criterion. In an illustrative example, the cluster load criterion may be determined as follows:

[0038] $$L = \sum_{i=1,\ldots n} (n_i * t_i)$$

[0039] wherein L denotes the cluster unused load, to be minimized by the batch scheduler, and n_i is the number of nodes that remain unused for a period of time t_i.

[0040] In an illustrative example, the batch scheduler may generate a plurality of combinations of applications of the plurality of applications to be executed by the cluster, determine the value of the cluster load criterion for each generated combination, and select the combination that is associated with the optimal (maximum or minimum) value of the cluster load criterion.

[0041] In various illustrative examples, the batch scheduler may determine a cluster configuration that satisfies additional constraints derived from various parameters of the node allocation request, such as the minimum number of nodes to be allocated to the application and/or the a waiting preference parameter indicating whether an additional waiting time is preferred over allocation of a lesser number of nodes than requested.

[0042] Responsive to determining the cluster configuration that optimizes the cluster load criterion, the batch scheduler may notify the application of the actual number of nodes that have been assigned to the application, as described in more details herein above.

[0043] Fig. 4 depicts a flowchart of an example method 400 for determining the application configuration in a cluster of nodes, in accordance with one or more aspects of the present disclosure. Method 400 may be performed by processing logic that may comprise hardware (e.g., circuitry, dedicated logic, programmable logic, microcode, etc.), software (such as operations being performed by a functional unit), firmware or a combination thereof. In certain implementations, method 400 is performed by a batch scheduler being executed by example processing systems 100 of Fig. 4.

[0044] Referring to Fig. 4, method 400 begins at block 410, where the processor implementing the method may receive a node allocation request. The node allocation request may specify the expected application and the requested number of nodes for executing the application. In certain implementations, the node allocation request may further specify various additional application execution parameters, as described in more details herein above.

[0045] At block 420, the processor may determine, in view of the node allocation request and a current load on the plurality of nodes, the application configuration for the requesting application, including the number of nodes to be allocated to the application. The application configuration may be chosen to optimize a certain cluster load criterion, as described in mode details herein above.

[0046] At block 430, the processor may store the application configuration, including the number of nodes, in the application configuration data structure, as described in mode details herein above.

[0047] At block 440, the processor may store the address of the application configuration data structure in the control register, as described in mode details herein above.

[0048] At block 450, the processor may cause the application to be executed according to the determined application configuration.

[0049] Implementations of the present disclosure are not limited to computer systems. Alternative implementations of the present disclosure may be used in other devices such as handheld devices and embedded applications. Some examples of handheld devices include cellular phones, Internet Protocol devices, digital cameras, personal digital assistants (PDAs), and handheld PCs. Embedded applications may include a micro controller, a digital signal processor (DSP), system on a chip, network computers (NetPC), set-top boxes, network hubs, wide area network (WAN) switches, or any other system that may perform one or more instructions in accordance with at least one implementation.

[0050] Referring again to Fig. 3, processor 102 includes one or more execution units 408 to implement an algorithm that is to perform at least one instruction. One implementation may be described in the context of a single processor desktop or server system, but alternative implementations may be included in a multiprocessor system. System 100 is an example of the 'hub' system architecture. Processing system 100 includes a processor 102 to process data signals. Processor 102, as one illustrative example, includes a complex instruction set computer (CISC) microprocessor, a reduced instruction set computing (RISC) microprocessor, a very long instruction word (VLIW) microprocessor, a processor implementing a combination of instruction sets, or any other processor device, such as a digital signal processor, for example. Processor 102 is coupled to a processor bus 410 that transmits data signals between the processor 102 and other components in the system 100. The elements of system 100 (e.g. graphics accelerator 412, memory controller hub 416, memory 420, I/O controller hub 430, wireless transceiver 426, flash BIOS 428, network controller 434, audio controller 436, serial expansion port 438, I/O controller 430, etc.) perform their conventional functions that are well known to those familiar with the art.

[0051] In one implementation, the processor 102 includes a Level 1 (L1) internal cache 404. Depending on the architecture, the processor 102 may have a single internal cache or multiple levels of internal caches. Other implementations include a combination of both internal and external caches depending on the particular implementation and needs. One or more of these caches may be set-associative and may allow address locking. As such, it may include an implementation whereby the cache controller implements the principles of locking and unlocking cache ways, as described herein. Register file 406 is to store different types of data in various registers including integer registers, floating point registers, vector registers, banked registers, shadow registers, checkpoint registers, status registers, and instruction pointer register.

[0052] Execution unit 408, including logic to perform integer and floating point operations, also resides in the processor 102. The processor 102, in one implementation, includes

a microcode (ucode) ROM to store microcode, which when executed, is to perform algorithms for certain macroinstructions or handle complex scenarios. Here, microcode is potentially updateable to handle logic bugs/fixes for processor 102. For one implementation, execution unit 408 includes logic to handle a packed instruction set 409. By including the packed instruction set 409 in the instruction set of a general-purpose processor 102, along with associated circuitry to execute the instructions, the operations used by many multimedia applications may be performed using packed data in a general-purpose processor 102. Thus, many multimedia applications are accelerated and executed more efficiently by using the full width of a processor's data bus for performing operations on packed data. This potentially eliminates the need to transfer smaller units of data across the processor's data bus to perform one or more operations, one data element at a time.

[0053] Alternate implementations of an execution unit 408 may also be used in micro controllers, embedded processors, graphics devices, DSPs, and other types of logic circuits. System 100 includes a memory 420. Memory 420 includes a dynamic random access memory (DRAM) device, a static random access memory (SRAM) device, flash memory device, or other memory device. Memory 420 stores instructions and/or data represented by data signals that are to be executed by the processor 102.

[0054] Memory controller hub (MCH) 416 is coupled to the processor bus 410 and memory 420. The processor 102 may communicate to the MCH 416 via a processor bus 410. The MCH 416 provides a high bandwidth memory path 418 to memory 420 for instruction and data storage and for storage of graphics commands, data and textures. The MCH 416 is to direct data signals between the processor 102, memory 420, and other components in the system 100 and to bridge the data signals between processor bus 410, memory 420, and system I/O 422. In some implementations, the system logic chip 416 may provide a graphics port for coupling to a graphics controller 412. The MCH 416 is coupled to memory 420 through a memory interface 418. The graphics card 412 is coupled to the MCH 416 through an Accelerated Graphics Port (AGP) interconnect 414. A cache may also be implemented in the Memory Controller Hub 416 to provide a faster response than memory 420. This cache may be set-associative and may allow the locking of addresses, as described herein.

[0055] System 100 may use a proprietary hub interface bus 422 to couple the MCH 416 to the I/O controller hub (ICH) 430. The ICH 430 provides direct connections to some I/O devices via a local I/O bus. The local I/O bus is a high-speed I/O bus for connecting peripherals to the memory 420, chipset, and processor 102. Some examples are the audio controller, firmware hub (flash BIOS) 428, wireless transceiver 426, data storage 424, legacy I/O controller containing user input and keyboard interfaces, a serial expansion port such as Universal Serial

Bus (USB), and a network controller 434. The data storage device 424 may comprise a hard disk drive, a floppy disk drive, a CD-ROM device, a flash memory device, or other mass storage device.

[0056] Fig. 5 illustrates various packed data type representations in processor registers according to one implementation of the present disclosure. Fig. 5 illustrates data types for a packed byte 510, a packed word 520, and a packed doubleword (dword) 530 for 128 bits wide operands. The packed byte format 510 of this example is 128 bits long and contains sixteen packed byte data elements. A byte is defined here as 8 bits of data. Information for each byte data element is stored in bit 7 through bit 0 for byte 0, bit 15 through bit 8 for byte 1, bit 23 through bit 16 for byte 2, and finally bit 120 through bit 127 for byte 15. Thus, all available bits are used in the register. This storage arrangement increases the storage efficiency of the processor. As well, with sixteen data elements accessed, one operation may now be performed on sixteen data elements in parallel.

[0057] Generally, a data element is an individual piece of data that is stored in a single register or memory location with other data elements of the same length. In packed data sequences relating to SSEx technology, the number of data elements stored in a XMM register is 128 bits divided by the length in bits of an individual data element. Similarly, in packed data sequences relating to MMX and SSE technology, the number of data elements stored in an MMX register is 64 bits divided by the length in bits of an individual data element. Although the data types illustrated in Fig. 5 are 128 bit long, implementations may also operate with 64 bit wide or other sized operands. The packed word format 520 of this example is 128 bits long and contains eight packed word data elements. Each packed word contains sixteen bits of information. The packed doubleword format 530 of Fig. 5 is 128 bits long and contains four packed doubleword data elements. Each packed doubleword data element contains thirty two bits of information. A packed quadword is 128 bits long and contains two packed quad-word data elements.

[0058] Fig. 6A is a block diagram illustrating an in-order pipeline and a register renaming stage, out-of-order issue/execution pipeline according to at least one implementation of the disclosure. Fig. 6B is a block diagram illustrating an in-order architecture core and a register renaming logic, out-of-order issue/execution logic to be included in a processor according to at least one implementation of the disclosure. The solid lined boxes in Fig. 6A illustrate the in-order pipeline, while the dashed lined boxes illustrates the register renaming, out-of-order issue/execution pipeline. Similarly, the solid lined boxes in Fig. 6B illustrate the in-order architecture logic, while the dashed lined boxes illustrates the register renaming logic and out-of-order issue/execution logic.

[0059] In Fig. 6A, a processor pipeline 600 includes a fetch stage 602, a length decode stage 604, a decode stage 606, an allocation stage 608, a renaming stage 610, a scheduling (also known as a dispatch or issue) stage 612, a register read/memory read stage 614, an execute stage 616, a write back/memory write stage 618, an exception handling stage 622, and a commit stage 624.

[0060] In Fig. 6B, arrows denote a coupling between two or more units and the direction of the arrow indicates a direction of data flow between those units. Fig. 6B shows processor core 690 including a front end unit 630 coupled to an execution engine unit 650, and both are coupled to a memory unit 670.

[0061] The core 690 may be a reduced instruction set computing (RISC) core, a complex instruction set computing (CISC) core, a very long instruction word (VLIW) core, or a hybrid or alternative core type. As yet another option, the core 690 may be a special-purpose core, such as, for example, a network or communication core, compression engine, graphics core, or the like.

[0062] The front end unit 630 includes a branch prediction unit 632 coupled to an instruction cache unit 634, which is coupled to an instruction translation lookaside buffer (TLB) 636, which is coupled to an instruction fetch unit 638, which is coupled to a decode unit 640. The decode unit or decoder may decode instructions, and generate as an output one or more micro-operations, micro-code entry points, microinstructions, other instructions, or other control signals, which are decoded from, or which otherwise reflect, or are derived from, the original instructions. The decoder may be implemented using various different mechanisms. Examples of suitable mechanisms include, but are not limited to, look-up tables, hardware implementations, programmable logic arrays (PLAs), microcode read only memories (ROMs), etc. The instruction cache unit 634 is further coupled to a level 2 (L2) cache unit 676 in the memory unit 670. The decode unit 640 is coupled to a rename/allocator unit 652 in the execution engine unit 650.

[0063] The execution engine unit 650 includes the rename/allocator unit 652 coupled to a retirement unit 654 and a set of one or more scheduler unit(s) 656. The scheduler unit(s) 656 represents any number of different schedulers, including reservations stations, central instruction window, etc. The scheduler unit(s) 656 is coupled to the physical register file(s) unit(s) 658. Each of the physical register file(s) units 658 represents one or more physical register files, different ones of which store one or more different data types, such as scalar integer, scalar floating point, packed integer, packed floating point, vector integer, vector floating point, status (e.g., an instruction pointer that is the address of the next instruction to be executed), etc. The physical register file(s) unit(s) 658 is overlapped by the retirement unit 654 to illustrate various ways in which register renaming and out-of-order execution may be implemented (e.g., using a reorder buffer(s) and a retirement register file(s), using a future file(s), a history buffer(s), and a

retirement register file(s); using a register maps and a pool of registers; etc.). Generally, the architectural registers are visible from the outside of the processor or from a programmer's perspective. The registers are not limited to any known particular type of circuit. Various different types of registers are suitable as long as they are capable of storing and providing data as described herein. Examples of suitable registers include, but are not limited to, dedicated physical registers, dynamically allocated physical registers using register renaming, combinations of dedicated and dynamically allocated physical registers, etc. The retirement unit 654 and the physical register file(s) unit(s) 658 are coupled to the execution cluster(s) 660. The execution cluster(s) 660 includes a set of one or more execution units 162 and a set of one or more memory access units 664. The execution units 662 may perform various operations (e.g., shifts, addition, subtraction, multiplication) and on various types of data (e.g., scalar floating point, packed integer, packed floating point, vector integer, vector floating point). While some implementations may include a number of execution units dedicated to specific functions or sets of functions, other implementations may include one execution unit or multiple execution units that all perform all functions. The scheduler unit(s) 656, physical register file(s) unit(s) 658, and execution cluster(s) 660 are shown as being possibly plural because certain implementations create separate pipelines for certain types of data/operations (e.g., a scalar integer pipeline, a scalar floating point/packed integer/packed floating point/vector integer/vector floating point pipeline, and/or a memory access pipeline that each have their own scheduler unit, physical register file(s) unit, and/or execution cluster – and in the case of a separate memory access pipeline, certain implementations are implemented in which the execution cluster of this pipeline has the memory access unit(s) 664). It should also be understood that where separate pipelines are used, one or more of these pipelines may be out-of-order issue/execution and the rest in-order.

[0064] The set of memory access units 664 is coupled to the memory unit 670, which includes a data TLB unit 672 coupled to a data cache unit 674 coupled to a level 2 (L2) cache unit 676. In one exemplary implementation, the memory access units 664 may include a load unit, a store address unit, and a store data unit, each of which is coupled to the data TLB unit 672 in the memory unit 670. The L2 cache unit 676 is coupled to one or more other levels of cache and eventually to a main memory. The L2 cache unit 676 may be set associative and may allow the locking of addresses, as described herein.

[0065] By way of example, the register renaming, out-of-order issue/execution core architecture may implement the pipeline 500 as follows: 1) the instruction fetch 638 performs the fetch and length decoding stages 602 and 604; 2) the decode unit 640 performs the decode stage 606; 3) the rename/allocator unit 652 performs the allocation stage 608 and renaming stage

610; 4) the scheduler unit(s) 656 performs the schedule stage 612; 5) the physical register file(s) unit(s) 658 and the memory unit 670 perform the register read/memory read stage 614; the execution cluster 660 perform the execute stage 616; 6) the memory unit 670 and the physical register file(s) unit(s) 658 perform the write back/memory write stage 618; 7) various units may be involved in the exception handling stage 622; and 8) the retirement unit 654 and the physical register file(s) unit(s) 658 perform the commit stage 624.

[0066] The core 690 may support one or more instructions sets (e.g., the x86 instruction set (with some extensions that have been added with newer versions); the MIPS instruction set of MIPS Technologies of Sunnyvale, CA; the ARM instruction set (with additional extensions such as NEON) of ARM Holdings of Sunnyvale, CA).

[0067] While register renaming is described in the context of out-of-order execution, it should be understood that register renaming may be used in an in-order architecture. While the illustrated implementation of the processor also includes a separate instruction and data cache units 634/674 and a shared L2 cache unit 676, alternative implementations may have a single internal cache for both instructions and data, such as, for example, a Level 1 (L1) internal cache, or multiple levels of internal cache. In some implementations, the system may include a combination of an internal cache and an external cache that is external to the core and/or the processor. Alternatively, all of the cache may be external to the core and/or the processor.

[0068] Fig. 7 is a block diagram of the micro-architecture for a processor 700 that includes logic circuits to perform instructions in accordance with one implementation of the present disclosure. In some implementations, an instruction in accordance with one implementation may be implemented to operate on data elements having sizes of byte, word, doubleword, quadword, etc., as well as datatypes, such as single and double precision integer and floating point datatypes. In one implementation the in-order front end 701 is the part of the processor 700 that fetches instructions to be executed and prepares them to be used later in the processor pipeline. The front end 701 may include several units. In one implementation, the instruction prefetcher 726 fetches instructions from memory and feeds them to an instruction decoder 728 which in turn decodes or interprets them. For example, in one implementation, the decoder decodes a received instruction into one or more operations called “micro-instructions” or “micro-operations” (also called micro op or uops) that the machine may execute. In other implementations, the decoder parses the instruction into an opcode and corresponding data and control fields that are used by the micro-architecture to perform operations in accordance with one implementation. In one implementation, the trace cache 730 takes decoded uops and assembles them into program ordered sequences or traces in the uop queue 734 for execution.

When the trace cache 730 encounters a complex instruction, the microcode ROM 732 provides the uops needed to complete the operation.

[0069] Some instructions are converted into a single micro-op, whereas others need several micro-ops to complete the full operation. In one implementation, if more than four micro-ops are needed to complete an instruction, the decoder 728 accesses the microcode ROM 732 to do the instruction. For one implementation, an instruction may be decoded into a small number of micro ops for processing at the instruction decoder 728. In another implementation, an instruction may be stored within the microcode ROM 732 should a number of micro-ops be needed to accomplish the operation. The trace cache 730 refers to an entry point programmable logic array (PLA) to determine a correct micro-instruction pointer for reading the micro-code sequences to complete one or more instructions in accordance with one implementation from the micro-code ROM 732. After the microcode ROM 732 finishes sequencing micro-ops for an instruction, the front end 701 of the machine resumes fetching micro-ops from the trace cache 730.

[0070] The out-of-order execution engine 703 is where the instructions are prepared for execution. The out-of-order execution logic has a number of buffers to smooth out and re-order the flow of instructions to optimize performance as they go down the pipeline and get scheduled for execution. The allocator logic allocates the machine buffers and resources that each uop needs in order to execute. The register renaming logic renames logic registers onto entries in a register file. The allocator also allocates an entry for each uop in one of the two uop queues, one for memory operations and one for non-memory operations, in front of the instruction schedulers: memory scheduler, fast scheduler 702, slow/general floating point scheduler 704, and simple floating point scheduler 706. The uop schedulers 702, 704, 706 determine when a uop is ready to execute based on the readiness of their dependent input register operand sources and the availability of the execution resources the uops need to complete their operation. The fast scheduler 702 of one implementation may schedule on each half of the main clock cycle while the other schedulers may schedule once per main processor clock cycle. The schedulers arbitrate for the dispatch ports to schedule uops for execution.

[0071] Register files 708, 710 sit between the schedulers 702, 704, 706, and the execution units 712, 714, 716, 718, 720, 722, 724 in the execution block 711. There is a separate register file 708, 710 for integer and floating point operations, respectively. Each register file 708, 710, of one implementation also includes a bypass network that may bypass or forward just completed results that have not yet been written into the register file to new dependent uops. The integer register file 708 and the floating point register file 710 are also capable of communicating data with the other. For one implementation, the integer register file 708 is split into two separate

register files, one register file for the low order 32 bits of data and a second register file for the high order 32 bits of data. The floating point register file 710 of one implementation has 128 bit wide entries because floating point instructions typically have operands from 64 to 128 bits in width.

[0072] The execution block 711 contains the execution units 712, 714, 716, 718, 720, 722, 724, where the instructions are actually executed. This section includes the register files 708, 710, that store the integer and floating point data operand values that the micro-instructions need to execute. The processor 700 of one implementation is comprised of a number of execution units: address generation unit (AGU) 712, AGU 714, fast ALU 716, fast ALU 718, slow ALU 720, floating point ALU 722, floating point move unit 724. For one implementation, the floating point execution blocks 722, 724, execute floating point, MMX, SIMD, and SSE, or other operations. The floating point ALU 722 of one implementation includes a 64 bit by 64 bit floating point divider to execute divide, square root, and remainder micro-ops. For implementations of the present disclosure, instructions involving a floating point value may be handled with the floating point hardware. In one implementation, the ALU operations go to the high-speed ALU execution units 716, 718. The fast ALUs 716, 718, of one implementation may execute fast operations with an effective latency of half a clock cycle. For one implementation, most complex integer operations go to the slow ALU 720 as the slow ALU 720 includes integer execution hardware for long latency type of operations, such as a multiplier, shifts, flag logic, and branch processing. Memory load/store operations are executed by the AGUs 712, 714. For one implementation, the integer ALUs 716, 718, 720 are described in the context of performing integer operations on 64 bit data operands. In alternative implementations, the ALUs 716, 718, 720 may be implemented to support a variety of data bits including 16, 32, 128, 756, etc. Similarly, the floating point units 722, 724 may be implemented to support a range of operands having bits of various widths. For one implementation, the floating point units 722, 724 may operate on 128 bits wide packed data operands in conjunction with SIMD and multimedia instructions.

[0073] In one implementation, the uops schedulers 702, 704, 706 dispatch dependent operations before the parent load has finished executing. As uops are speculatively scheduled and executed in processor 700, the processor 700 also includes logic to handle memory misses. If a data load misses in the data cache, there may be dependent operations in flight in the pipeline that have left the scheduler with temporarily incorrect data. A replay mechanism tracks and re-executes instructions that use incorrect data. The dependent operations should be replayed and the independent ones are allowed to complete. The schedulers and replay mechanism of one

implementation of a processor are also designed to catch instruction sequences for text string comparison operations.

[0074] The term "registers" may refer to the on-board processor storage locations that are used as part of instructions to identify operands. In other words, registers may be those that are usable from the outside of the processor (from a programmer's perspective). However, the registers of an implementation should not be limited in meaning to a particular type of circuit. Rather, a register of an implementation is capable of storing and providing data, and performing the functions described herein. The registers described herein may be implemented by circuitry within a processor using any number of different techniques, such as dedicated physical registers, dynamically allocated physical registers using register renaming, combinations of dedicated and dynamically allocated physical registers, etc. In one implementation, integer registers store thirty-two bit integer data. A register file of one implementation also contains eight multimedia SIMD registers for packed data. For the discussions below, the registers are understood to be data registers designed to hold packed data, such as 64 bits wide MMX registers (also referred to as 'mm' registers in some instances) in microprocessors enabled with the MMX™ technology from Intel Corporation of Santa Clara, California. These MMX registers, available in both integer and floating point forms, may operate with packed data elements that accompany SIMD and SSE instructions. Similarly, 128 bits wide XMM registers relating to SSE2, SSE3, SSE4, or beyond (referred to generically as "SSEx") technology may also be used to hold such packed data operands. In one implementation, in storing packed data and integer data, the registers do not need to differentiate between the two data types. In one implementation, integer and floating point are either contained in the same register file or different register files. Furthermore, in one implementation, floating point and integer data may be stored in different registers or the same registers.

[0075] Fig. 8 is a block diagram of a single core processor and a multicore processor 800 with integrated memory controller and graphics according to implementations of the disclosure. The solid lined boxes in Fig. 8 illustrate a processor 800 with a single core 802A, a system agent 810, a set of one or more bus controller units 816, while the addition of the dashed lined boxes illustrates an alternative processor 800 with multiple cores 802A-N, a set of one or more integrated memory controller unit(s) 814 in the system agent unit 810, and an integrated graphics logic 808.

[0076] The memory hierarchy includes one or more levels of cache within the cores, a set or one or more shared cache units 806, and external memory (not shown) coupled to the set of integrated memory controller units 814. The set of shared cache units 806 may include one or more mid-level caches, such as level 2 (L2), level 3 (L3), level 4 (L4), or other levels of cache, a

last level cache (LLC), and/or combinations thereof. This set of shared cache units 806 may be set associative and may allow the locking of addresses, as described herein. While in one implementation a ring based interconnect unit 812 interconnects the integrated graphics logic 808, the set of shared cache units 806, and the system agent unit 810, alternative implementations may use any number of well-known techniques for interconnecting such units.

[0077] In some implementations, one or more of the cores 802A-N are capable of multi-threading.

[0078] The system agent 810 includes those components coordinating and operating cores 802A-N. The system agent unit 810 may include for example a power control unit (PCU) and a display unit. The PCU may be or include logic and components needed for regulating the power state of the cores 802A-N and the integrated graphics logic 808. The display unit is for driving one or more externally connected displays.

[0079] The cores 802A-N may be homogenous or heterogeneous in terms of architecture and/or instruction set. For example, some of the cores 802A-N may be in order while others are out-of-order. As another example, two or more of the cores 802A-N may be capable of execution the same instruction set, while others may be capable of executing a subset of that instruction set or a different instruction set. As a further example, the cores may be different architecture.

[0080] The processor may include one or more different general-purpose processors, such as a Core™ i3, i5, i7, 2 Duo and Quad, Xeon™, Itanium™, Atom™, XScale™ or StrongARM™ processor, which are available from Intel Corporation, of Santa Clara, Calif. For example, one core may be a Core i7™ core while another core of the processor may be an Atom™ core. Alternatively, the processor may be from another company, such as ARM Holdings, Ltd, MIPS, etc. The processor may be a special-purpose processor, such as, for example, a network or communication processor, compression engine, graphics processor, co-processor, embedded processor, or the like. The processor may be implemented on one or more chips. The processor 800 may be a part of and/or may be implemented on one or more substrates using any of a number of process technologies, such as, for example, BiCMOS, CMOS, or NMOS.

[0081] Referring now to Fig. 9, shown is a block diagram of a SoC 900 in accordance with an implementation of the present disclosure. Similar elements in Fig. 9 bear like reference numerals. Also, dashed lined boxes are features on more advanced SoCs. In Fig. 9, an interconnect unit(s) 902 is coupled to: an application processor 910 which includes a set of one or more cores 902A-N and shared cache unit(s) 906; a system agent unit 910; a bus controller unit(s) 916; an integrated memory controller unit(s) 914; a set or one or more media processors 920 which may include integrated graphics logic 908, an image processor 924 for providing still

and/or video camera functionality, an audio processor 926 for providing hardware audio acceleration, and a video processor 928 for providing video encode/decode acceleration; an static random access memory (SRAM) unit 930; a direct memory access (DMA) unit 932; and a display unit 940 for coupling to one or more external displays.

[0082] Implementations may be implemented in many different system types. Referring now to Fig. 10, shown is a block diagram of a multiprocessor system 1000 in accordance with some implementations. As shown in Fig. 10, multiprocessor system 1000 is a point-to-point interconnect system, and includes a first processor 1070 and a second processor 1080 coupled via a point-to-point interconnect 1050. As shown in Fig. 10, each of processors 1070 and 1080 may be multicore processors, including first and second processor cores (i.e., processor cores 1074a and 1074b and processor cores 1084a and 1084b), although potentially many more cores may be present in the processors. The processors each may include hybrid write mode logics in accordance with an implementation of the present.

[0083] While shown with two processors 1070, 1080, it is to be understood that the scope of the present disclosure is not so limited. In other implementations, one or more additional processors may be present in a given processor.

[0084] Processors 1070 and 1080 are shown including integrated memory controller units 8102 and 8102, respectively. Processor 1070 also includes as part of its bus controller units point-to-point (P-P) interfaces 1076 and 1078; similarly, second processor 1080 includes P-P interfaces 1086 and 1088. Processors 1070, 1080 may exchange information via a point-to-point (P-P) interface 1050 using P-P interface circuits 1078, 1088. As shown in Fig. 10, IMCs 1072 and 1082 couple the processors to respective memories, namely a memory 1032 and a memory 1034, which may be portions of main memory locally attached to the respective processors.

[0085] Processors 1070, 1080 may each exchange information with a chipset 1090 via individual P-P interfaces 1052, 1054 using point to point interface circuits 1076, 1094, 1086, 1098. Chipset 1090 may also exchange information with a high-performance graphics circuit 1038 via a high-performance graphics interface 1039.

[0086] A shared cache (not shown) may be included in either processor or outside of both processors, yet connected with the processors via P-P interconnect, such that either or both processors' local cache information may be stored in the shared cache if a processor is placed into a low power mode.

[0087] Chipset 1090 may be coupled to a first bus 1016 via an interface 1096. In one implementation, first bus 1016 may be a Peripheral Component Interconnect (PCI) bus, or a bus such as a PCI Express bus or another third generation I/O interconnect bus, although the scope of the present disclosure is not so limited.

[0088] As shown in Fig. 10, various I/O devices 1014 may be coupled to first bus 1016, along with a bus bridge 1018 which couples first bus 1016 to a second bus 1020. In one implementation, second bus 1020 may be a low pin count (LPC) bus. Various devices may be coupled to second bus 1020 including, for example, a keyboard and/or mouse 1022, communication devices 1027 and a storage unit 1028 such as a disk drive or other mass storage device which may include instructions/code and data 1030, in one implementation. Further, an audio I/O 1024 may be coupled to second bus 1020. Note that other architectures are possible. For example, instead of the point-to-point architecture of Fig. 10, a system may implement a multi-drop bus or other such architecture.

[0089] Turning next to Fig. 11, an implementation of a system on-chip (SOC) design in accordance with implementations of the disclosure is depicted. As an illustrative example, SOC 1100 is included in user equipment (UE). In one implementation, UE refers to any device to be used by an end-user to communicate, such as a hand-held phone, smartphone, tablet, ultra-thin notebook, notebook with broadband adapter, or any other similar communication device. A UE may connect to a base station or node, which may correspond in nature to a mobile station (MS) in a GSM network.

[0090] Here, SOC 1100 includes 2 cores—1106 and 1107. Similar to the discussion above, cores 1106 and 1107 may conform to an Instruction Set Architecture, such as a processor having the Intel® Architecture Core™, an Advanced Micro Devices, Inc. (AMD) processor, a MIPS-based processor, an ARM-based processor design, or a customer thereof, as well as their licensees or adopters. Cores 1106 and 1107 are coupled to cache control 1108 that is associated with bus interface unit 1109 and L2 cache 1110 to communicate with other parts of system 1100. Interconnect 1111 includes an on-chip interconnect, such as an IOSF, AMBA, or other interconnects discussed above, which may implement one or more aspects of the described disclosure.

[0091] Interconnect 1111 provides communication channels to the other components, such as a Subscriber Identity Module (SIM) 1130 to interface with a SIM card, a boot rom 1135 to hold boot code for execution by cores 1106 and 1107 to initialize and boot SOC 1100, a SDRAM controller 1140 to interface with external memory (e.g. DRAM 1160), a flash controller 1145 to interface with persistent or non-volatile memory (e.g. Flash 1165), a peripheral control 1150 (e.g. Serial Peripheral Interface) to interface with peripherals, video codecs 1120 and Video interface 1125 to display and receive input (e.g. touch enabled input), GPU 1115 to perform graphics related computations, etc. Any of these interfaces may incorporate aspects of the implementations described herein.

[0092] In addition, the system illustrates peripherals for communication, such as a Bluetooth module 1170, modem 1175 (e.g., 3G, 4G, Long Term Evolution (LTE), LTE-Advanced, etc.), GPS 1180, Wi-Fi 1185, Zigbee (not shown), and Z-Wave (not shown). Note as stated above, a UE includes a radio for communication. As a result, these peripheral communication modules may not all be included. However, in a UE some form of a radio for external communication should be included.

[0093] Fig. 12 illustrates a diagrammatic representation of a machine in the example form of a computing system 1200 within which a set of instructions, for causing the machine to perform any one or more of the methodologies discussed herein, may be executed. In alternative implementations, the machine may be connected (e.g., networked) to other machines in a LAN, an intranet, an extranet, or the Internet. The machine may operate in the capacity of a server or a client device in a client-server network environment, or as a peer machine in a peer-to-peer (or distributed) network environment. The machine may be a personal computer (PC), a tablet PC, a set-top box (STB), a Personal Digital Assistant (PDA), a game console, a cellular telephone, a digital camera, a handheld PC, a web appliance, a server, a network router, switch or bridge, micro controller, a digital signal processor (DSP), system on a chip, network computers (NetPC), network hubs, wide area network (WAN) switches, or any machine capable of executing a set of instructions (sequential or otherwise) that specify actions to be taken by that machine. Further, while only a single machine is illustrated for the system architecture 100, the term "machine" shall also be taken to include any collection of machines that individually or jointly execute a set (or multiple sets) of instructions to perform any one or more of the methodologies discussed herein. Implementations are not limited to computer systems.

[0094] The computing system 1200 includes a processing device 1202, main memory 1204 (e.g., read-only memory (ROM), flash memory, dynamic random access memory (DRAM) (such as synchronous DRAM (SDRAM) or DRAM (RDRAM), etc.), a static memory 1206 (e.g., flash memory, static random access memory (SRAM), etc.), and a data storage device 1216, which communicate with each other via a bus 1208.

[0095] Processing device 1202 represents one or more general-purpose processing devices such as a microprocessor, central processing unit, or the like. More particularly, the processing device may be complex instruction set computing (CISC) microprocessor, reduced instruction set computer (RISC) microprocessor, very long instruction word (VLIW) microprocessor, or processor implementing other instruction sets, or processors implementing a combination of instruction sets. Processing device 1202 may also be one or more special-purpose processing devices such as an application specific integrated circuit (ASIC), a field programmable gate array (FPGA), a digital signal processor (DSP), network processor, or the

like. In one implementation, processing device 1202 may include one or processing cores. The processing device 1202 is configured to execute the processing logic 1226 for performing the operations discussed herein. In one implementation, processing device 1202 may be part of the system architecture 100 of Fig. 1A or SoC 190 of Fig. 1B. Alternatively, the computing system 1200 may include other components as described herein. It should be understood that the core may support multithreading (executing two or more parallel sets of operations or threads), and may do so in a variety of ways including time sliced multithreading, simultaneous multithreading (where a single physical core provides a logical core for each of the threads that physical core is simultaneously multithreading), or a combination thereof (e.g., time sliced fetching and decoding and simultaneous multithreading thereafter such as in the Intel® Hyperthreading technology).

[0096] Computing system 1200 is representative of processing systems based on the PENTIUM III™, PENTIUM 4™, Celeron™, Xeon™, Itanium, XScale™, StrongARM™, Core™, Core 2™, Atom™, and/or Intel® Architecture Core™, such as an i3, i5, i7 microprocessors available from Intel Corporation of Santa Clara, California, although other systems (including PCs having other microprocessors, engineering workstations, set-top boxes and the like) may also be used. However, understand that other low power processors such as available from Advanced Micro Devices, Inc. (AMD) of Sunnyvale, CA, a MIPS-based design from MIPS Technologies, Inc. of Sunnyvale, CA, an ARM-based design licensed from ARM Holdings, Ltd. or customer thereof, or their licensees or adopters may instead be present in other implementations such as an Apple A5/A6 processor, a Qualcomm Snapdragon processor, or TI OMAP processor. In one implementation, processing device 1202 executes a version of the WINDOWS™ operating system available from Microsoft Corporation of Redmond, Washington, although other operating systems (OS X, UNIX, Linux, Android, iOS, Symbian, for example), embedded software, and/or graphical user interfaces, may also be used. Thus, implementations of the present disclosure are not limited to any specific combination of hardware circuitry and software. One implementation may be described in the context of a single processor desktop or server system, but alternative implementations may be included in a multiprocessor system. Computing system 1200 may be an example of a 'hub' system architecture.

[0097] The computing system 1200 may further include a network interface device 1222 communicably coupled to a network 1218. The computing system 1200 also may include a display device 1210 (e.g., a liquid crystal display (LCD) or a cathode ray tube (CRT)), an alphanumeric input device 1212 (e.g., a keyboard), a cursor control device 1214 (e.g., a mouse), a signal generation device 1220 (e.g., a speaker), or other peripheral devices. Furthermore, computing system 1200 may include a graphics processing unit (not illustrated), a video

processing unit (not illustrated) and an audio processing unit (not illustrated). In another implementation, the computing system 1200 may include a chipset (not illustrated), which refers to a group of integrated circuits, or chips, that are designed to work with the processing device 1202 and controls communications between the processing device 1202 and external devices. For example, the chipset may be a set of chips on a motherboard that links the processing device 1202 to very high-speed devices, such as main memory 1204 and graphic controllers, as well as linking the processing device 1202 to lower-speed peripheral buses of peripherals, such as USB, PCI or ISA buses.

[0098] The data storage device 1216 may include a computer-readable storage medium 1224 on which is stored instructions 1226 embodying any one or more of the methodologies of functions described herein. The instructions 1226 may also reside, completely or at least partially, within the main memory 1204 as instructions 1226 and/or within the processing device 1202 as processing logic 1226 during execution thereof by the computing system 1200; the main memory 1204 and the processing device 1202 also constituting computer-readable storage media.

[0099] The computer-readable storage medium 1224 may also be used to store instructions 1226 utilizing the processing device 1202, such as described with respect to Fig. 1A, and/or a software library containing methods that call the above applications. While the computer-readable storage medium 1224 is shown in an example implementation to be a single medium, the term "computer-readable storage medium" should be taken to include a single medium or multiple media (e.g., a centralized or distributed database, and/or associated caches and servers) that store the one or more sets of instructions. The term "computer-readable storage medium" shall also be taken to include any medium that is capable of storing, encoding or carrying a set of instruction for execution by the machine and that cause the machine to perform any one or more of the methodologies of the present implementations. The term "computer-readable storage medium" shall accordingly be taken to include, but not be limited to, solid-state memories, and optical and magnetic media.

[00100] While the present disclosure has been described with respect to a limited number of implementations, those skilled in the art will appreciate numerous modifications and variations therefrom. It is intended that the appended claims cover all such modifications and variations as fall within the true spirit and scope of this present disclosure.

[00101] In the description herein, numerous specific details are set forth, such as examples of specific types of processors and system configurations, specific hardware structures, specific architectural and micro architectural details, specific register configurations, specific instruction types, specific system components, specific measurements/heights, specific processor pipeline

stages and operation etc. in order to provide a thorough understanding of the present disclosure. It will be apparent, however, to one skilled in the art that these specific details need not be employed to practice the present disclosure. In other instances, well known components or methods, such as specific and alternative processor architectures, specific logic circuits/code for described algorithms, specific firmware code, specific interconnect operation, specific logic configurations, specific manufacturing techniques and materials, specific compiler implementations, specific expression of algorithms in code, specific power down and gating techniques/logic and other specific operational details of computer system have not been described in detail in order to avoid unnecessarily obscuring the present disclosure.

[00102] The implementations are described with reference to hybrid-threading in specific integrated circuits, such as in computing platforms or microprocessors. The implementations may also be applicable to other types of integrated circuits and programmable logic devices. For example, the disclosed implementations are not limited to desktop computer systems or portable computers, such as the Intel® Ultrabooks™ computers. And may be also used in other devices, such as handheld devices, tablets, other thin notebooks, systems on a chip (SOC) devices, and embedded applications. Some examples of handheld devices include cellular phones, Internet protocol devices, digital cameras, personal digital assistants (PDAs), and handheld PCs. Embedded applications typically include a microcontroller, a digital signal processor (DSP), a system on a chip, network computers (NetPC), set-top boxes, network hubs, wide area network (WAN) switches, or any other system that may perform the functions and operations taught below. It is described that the system may be any kind of computer or embedded system. The disclosed implementations may especially be used for low-end devices, like wearable devices (e.g., watches), electronic implants, sensory and control infrastructure devices, controllers, supervisory control and data acquisition (SCADA) systems, or the like. Moreover, the apparatuses, methods, and systems described herein are not limited to physical computing devices, but may also relate to software optimizations for energy conservation and efficiency. As will become readily apparent in the description below, the implementations of methods, apparatuses, and systems described herein (whether in reference to hardware, firmware, software, or a combination thereof) are vital to a 'green technology' future balanced with performance considerations.

[00103] Although the implementations herein are described with reference to a processor, other implementations are applicable to other types of integrated circuits and logic devices. Similar techniques and teachings of implementations of the present disclosure may be applied to other types of circuits or semiconductor devices that may benefit from higher pipeline throughput and improved performance. The teachings of implementations of the present

disclosure are applicable to any processor or machine that performs data manipulations. However, the present disclosure is not limited to processors or machines that perform 512 bit, 256 bit, 128 bit, 64 bit, 32 bit, or 16 bit data operations and may be applied to any processor and machine in which manipulation or management of data is performed. In addition, the description herein provides examples, and the accompanying drawings show various examples for the purposes of illustration. However, these examples should not be construed in a limiting sense as they are merely intended to provide examples of implementations of the present disclosure rather than to provide an exhaustive list of all possible implementations of implementations of the present disclosure.

[00104] Although the below examples describe instruction handling and distribution in the context of execution units and logic circuits, other implementations of the present disclosure may be accomplished by way of a data or instructions stored on a machine-readable, tangible medium, which when performed by a machine cause the machine to perform functions consistent with at least one implementation of the disclosure. In one implementation, functions associated with implementations of the present disclosure are embodied in machine-executable instructions. The instructions may be used to cause a general-purpose or special-purpose processor that is programmed with the instructions to perform the steps of the present disclosure. Implementations of the present disclosure may be provided as a computer program product or software which may include a machine or computer-readable medium having stored thereon instructions which may be used to program a computer (or other electronic devices) to perform one or more operations according to implementations of the present disclosure. Alternatively, operations of implementations of the present disclosure might be performed by specific hardware components that contain fixed-function logic for performing the operations, or by any combination of programmed computer components and fixed-function hardware components.

[00105] Instructions used to program logic to perform implementations of the disclosure may be stored within a memory in the system, such as DRAM, cache, flash memory, or other storage. Furthermore, the instructions may be distributed via a network or by way of other computer readable media. Thus a machine-readable medium may include any mechanism for storing or transmitting information in a form readable by a machine (e.g., a computer), but is not limited to, floppy diskettes, optical disks, Compact Disc, Read-Only Memory (CD-ROMs), and magneto-optical disks, Read-Only Memory (ROMs), Random Access Memory (RAM), Erasable Programmable Read-Only Memory (EPROM), Electrically Erasable Programmable Read-Only Memory (EEPROM), magnetic or optical cards, flash memory, or a tangible, machine-readable storage used in the transmission of information over the Internet via electrical, optical, acoustical or other forms of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.).

Accordingly, the computer-readable medium includes any type of tangible machine-readable medium suitable for storing or transmitting electronic instructions or information in a form readable by a machine (e.g., a computer).

[00106] A design may go through various stages, from creation to simulation to fabrication. Data representing a design may represent the design in a number of manners. First, as is useful in simulations, the hardware may be represented using a hardware description language or another functional description language. Additionally, a circuit level model with logic and/or transistor gates may be produced at some stages of the design process. Furthermore, most designs, at some stage, reach a level of data representing the physical placement of various devices in the hardware model. In the case where conventional semiconductor fabrication techniques are used, the data representing the hardware model may be the data specifying the presence or absence of various features on different mask layers for masks used to produce the integrated circuit. In any representation of the design, the data may be stored in any form of a machine readable medium. A memory or a magnetic or optical storage such as a disc may be the machine readable medium to store information transmitted via optical or electrical wave modulated or otherwise generated to transmit such information. When an electrical carrier wave indicating or carrying the code or design is transmitted, to the extent that copying, buffering, or re-transmission of the electrical signal is performed, a new copy is made. Thus, a communication provider or a network provider may store on a tangible, machine-readable medium, at least temporarily, an article, such as information encoded into a carrier wave, embodying techniques of implementations of the present disclosure.

[00107] A module as used herein refers to any combination of hardware, software, and/or firmware. As an example, a module includes hardware, such as a micro-controller, associated with a non-transitory medium to store code adapted to be executed by the micro-controller. Therefore, reference to a module, in one implementation, refers to the hardware, which is specifically configured to recognize and/or execute the code to be held on a non-transitory medium. Furthermore, in another implementation, use of a module refers to the non-transitory medium including the code, which is specifically adapted to be executed by the microcontroller to perform predetermined operations. And as may be inferred, in yet another implementation, the term module (in this example) may refer to the combination of the microcontroller and the non-transitory medium. Often module boundaries that are illustrated as separate commonly vary and potentially overlap. For example, a first and a second module may share hardware, software, firmware, or a combination thereof, while potentially retaining some independent hardware, software, or firmware. In one implementation, use of the term logic includes hardware, such as transistors, registers, or other hardware, such as programmable logic devices.

[00108] Use of the phrase 'configured to,' in one implementation, refers to arranging, putting together, manufacturing, offering to sell, importing and/or designing an apparatus, hardware, logic, or element to perform a designated or determined task. In this example, an apparatus or element thereof that is not operating is still 'configured to' perform a designated task if it is designed, coupled, and/or interconnected to perform said designated task. As a purely illustrative example, a logic gate may provide a 0 or a 1 during operation. But a logic gate 'configured to' provide an enable signal to a clock does not include every potential logic gate that may provide a 1 or 0. Instead, the logic gate is one coupled in some manner that during operation the 1 or 0 output is to enable the clock. Note once again that use of the term 'configured to' does not require operation, but instead focus on the latent state of an apparatus, hardware, and/or element, where in the latent state the apparatus, hardware, and/or element is designed to perform a particular task when the apparatus, hardware, and/or element is operating.

[00109] Furthermore, use of the phrases 'to,' 'capable of/to,' and or 'operable to,' in one implementation, refers to some apparatus, logic, hardware, and/or element designed in such a way to enable use of the apparatus, logic, hardware, and/or element in a specified manner. Note as above that use of to, capable to, or operable to, in one implementation, refers to the latent state of an apparatus, logic, hardware, and/or element, where the apparatus, logic, hardware, and/or element is not operating but is designed in such a manner to enable use of an apparatus in a specified manner.

[00110] A value, as used herein, includes any known representation of a number, a state, a logical state, or a binary logical state. Often, the use of logic levels, logic values, or logical values is also referred to as 1's and 0's, which simply represents binary logic states. For example, a 1 refers to a high logic level and 0 refers to a low logic level. In one implementation, a storage cell, such as a transistor or flash cell, may be capable of holding a single logical value or multiple logical values. However, other representations of values in computer systems have been used. For example the decimal number ten may also be represented as a binary value of 1010 and a hexadecimal letter A. Hexadecimal values may also be represented with a prefix, such as "0x." Therefore, a value includes any representation of information capable of being held in a computer system.

[00111] Moreover, states may be represented by values or portions of values. As an example, a first value, such as a logical one, may represent a default or initial state, while a second value, such as a logical zero, may represent a non-default state. In addition, the terms reset and set, in one implementation, refer to a default and an updated value or state, respectively. For example, a default value potentially includes a high logical value, i.e. reset,

while an updated value potentially includes a low logical value, i.e. set. Note that any combination of values may be utilized to represent any number of states.

[00112] The implementations of methods, hardware, software, firmware or code set forth above may be implemented via instructions or code stored on a machine-accessible, machine readable, computer accessible, or computer readable medium which are executable by a processing element. A non-transitory machine-accessible/readable medium includes any mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine, such as a computer or electronic system. For example, a non-transitory machine-accessible medium includes random-access memory (RAM), such as static RAM (SRAM) or dynamic RAM (DRAM); ROM; magnetic or optical storage medium; flash memory devices; electrical storage devices; optical storage devices; acoustical storage devices; other form of storage devices for holding information received from transitory (propagated) signals (e.g., carrier waves, infrared signals, digital signals); etc., which are to be distinguished from the non-transitory mediums that may receive information there from.

[00113] Reference throughout this specification to "one implementation" or "an implementation" means that a particular feature, structure, or characteristic described in connection with the implementation is included in at least one implementation of the present disclosure. Thus, the appearances of the phrases "in one implementation" or "in an implementation" in various places throughout this specification are not necessarily all referring to the same implementation. Furthermore, the particular features, structures, or characteristics may be combined in any suitable manner in one or more implementations.

[00114] In the present specification, a detailed description has been given with reference to specific example implementations. It will, however, be evident that various modifications and changes may be made thereto without departing from the broader spirit and scope of the disclosure as set forth in the appended claims. The specification and drawings are, accordingly, to be regarded in an illustrative sense rather than a restrictive sense. Furthermore, the foregoing use of implementation and other exemplarily language does not necessarily refer to the same implementation or the same example, but may refer to different and distinct implementations, as well as potentially the same implementation.

[00115] The following examples pertain to further implementations.

[00116] Example 1 is a processing system, comprising: a control register; and a processing core, communicatively coupled to the control register; wherein the processing core is configured to: receive a node allocation request specifying an expected running time of an application and a requested number of nodes of a cluster of nodes; determine, in view of the node allocation request and a current load on the plurality of nodes, an actual number of nodes to be allocated to

the application, wherein the actual number of nodes to be allocated to the application optimizes a cluster load criterion; and notify, using the control register, the application of the actual number of nodes.

[00117] Example 2 is the processing system of Example 1, wherein notifying the application of the actual number of nodes comprises: storing the actual number of nodes in a memory data structure that is employed to store a configuration of nodes associated with the application; and storing an address of the memory data structure in the control register.

[00118] Example 3 is the processing system of Example 2, wherein storing the address of the memory data structure in the control register comprises obtaining a lock of the control register.

[00119] Example 4 is the processing system of Example 1, wherein the processing core is further configured to: cause the application to be executed using the actual number of nodes.

[00120] Example 5 is the processing system of Example 1, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any currently running application within a certain time period.

[00121] Example 6 is the processing system of Example 1, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any running application within a certain time period, multiplied by the time period.

[00122] Example 7 is the processing system of Example 1, wherein determining the actual number of nodes further comprises: generating a plurality of combinations of applications to be scheduled; determining a value of the cluster load criterion for each generated combination; and selecting a combination that is associated with an optimal value of the cluster load criterion.

[00123] Example 8 is the processing system of Example 1, wherein the processing system is implemented as a System-on-Chip (SoC).

[00124] Example 9 is a method, comprising: receiving, by a processing device, a node allocation request specifying an expected running time of an application and a requested number of nodes of a cluster of nodes; determining, in view of the node allocation request and a current load on the plurality of nodes, an actual number of nodes to be allocated to the application, wherein the actual number of nodes to be allocated to the application optimizes a cluster load criterion; and notifying, using a control register of the processing system, the application of the actual number of nodes.

[00125] Example 10 is the method of Example 9, wherein notifying the application of the actual number of nodes comprises: storing the actual number of nodes in a memory data structure that is employed to store a configuration of nodes associated with the application; and storing an address of the memory data structure in the control register.

[00126] Example 11 is the method of Example 10, wherein storing the address of the memory data structure in the control register comprises obtaining a lock of the control register.

[00127] Example 12 is the method of Example 9, further comprising: causing the application to be executed using the actual number of nodes.

[00128] Example 13 is the method of Example 9, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any currently running application within a certain time period.

[00129] Example 14 is the method of Example 9, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any running application within a certain time period, multiplied by the time period.

[00130] Example 15 is the method of Example 9, wherein determining the actual number of nodes further comprises: generating a plurality of combinations of applications to be scheduled; determining a value of the cluster load criterion for each generated combination; and selecting a combination that is associated with an optimal value of the cluster load criterion.

[00131] Example 16 is an apparatus comprising: a memory; and a processing device coupled to the memory, the processing device to perform the method of any of examples 9-15.

[00132] Example 17 is a non-transitory computer-readable storage medium comprising executable instructions that, when executed by a processing system, cause the processing system to perform operations comprising: receiving a node allocation request specifying an expected running time of an application and a requested number of nodes of a cluster of nodes; determining, in view of the node allocation request and a current load on the plurality of nodes, an actual number of nodes to be allocated to the application, wherein the actual number of nodes to be allocated to the application optimizes a cluster load criterion; and notifying, using a control register of the processing system, the application of the actual number of nodes.

[00133] Example 18 is the non-transitory computer-readable storage medium of Example 17, wherein notifying the application of the actual number of nodes comprises: storing the actual number of nodes in a memory data structure that is employed to store a configuration of nodes associated with the application; and storing an address of the memory data structure in the control register.

[00134] Example 19 is the non-transitory computer-readable storage medium of Example 18, wherein storing the address of the memory data structure in the control register comprises obtaining a lock of the control register.

[00135] Example 20 is the non-transitory computer-readable storage medium of Example 17, further comprising executable instructions causing the processing system to: cause the application to be executed using the actual number of nodes.

[00136] Example 21 is the non-transitory computer-readable storage medium of Example 17, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any currently running application within a certain time period.

[00137] Example 22 is the non-transitory computer-readable storage medium of Example 17, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any running application within a certain time period, multiplied by the time period.

[00138] Example 23 is the non-transitory computer-readable storage medium of Example 17, wherein determining the actual number of nodes further comprises: generating a plurality of combinations of applications to be scheduled; determining a value of the cluster load criterion for each generated combination; and selecting a combination that is associated with an optimal value of the cluster load criterion.

[00139] Example 24 is a high-performance computing system, comprising: a control node; and a compute cluster comprising a plurality of compute nodes; wherein the control node is configured to: receive a compute node allocation request specifying an expected running time of an application and a requested number of compute nodes of the compute cluster; determine, in view of the compute node allocation request and a current load on the plurality of compute nodes, an actual number of compute nodes to be allocated to the application, wherein the actual number of compute nodes to be allocated to the application optimizes a cluster load criterion; and notify the application of the actual number of compute nodes.

[00140] Example 25 is the high-performance computing system of Example 24, wherein the control node is further configured to: cause the application to be executed using the actual number of nodes.

[00141] Example 26 is the high-performance computing system of Example 24, wherein notifying the application of the actual number of nodes comprises: storing the actual number of nodes in a memory data structure that is employed to store a configuration of nodes associated with the application; and storing an address of the memory data structure in the control register.

[00142] Example 27 is the high-performance computing system of Example 24, wherein storing the address of the memory data structure in the control register comprises obtaining a lock of the control register.

[00143] Example 28 is the high-performance computing system of Example 24, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any running application within a certain time period.

[00144] Example 29 is the high-performance computing system of Example 24, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any running application within a certain time period, multiplied by the time period.

[00145] Example 30 is the high-performance computing system of Example 24, wherein determining the actual number of nodes further comprises: generating a plurality of combinations of applications to be scheduled; determining a value of the cluster load criterion for each generated combination; and selecting a combination that is associated with an optimal value of the cluster load criterion.

[00146] Some portions of the detailed description are presented in terms of algorithms and symbolic representations of operations on data bits within a computer memory. These algorithmic descriptions and representations are the means used by those skilled in the data processing arts to most effectively convey the substance of their work to others skilled in the art. An algorithm is here and generally, conceived to be a self-consistent sequence of operations leading to a desired result. The operations are those requiring physical manipulations of physical quantities. Usually, though not necessarily, these quantities take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared and otherwise manipulated. It has proven convenient at times, principally for reasons of common usage, to refer to these signals as bits, values, elements, symbols, characters, terms, numbers or the like. The blocks described herein may be hardware, software, firmware or a combination thereof.

[00147] It should be borne in mind, however, that all of these and similar terms are to be associated with the appropriate physical quantities and are merely convenient labels applied to these quantities. Unless specifically stated otherwise as apparent from the above discussion, it is appreciated that throughout the description, discussions utilizing terms such as "defining," "receiving," "determining," "issuing," "linking," "associating," "obtaining," "authenticating," "prohibiting," "executing," "requesting," "communicating," "monitoring," "calculating," or the like, refer to the actions and processes of a computing system, or similar electronic computing device, that manipulates and transforms data represented as physical (e.g., electronic) quantities within the computing system's registers and memories into other data similarly represented as physical quantities within the computing system memories or registers or other such information storage, transmission or display devices.

[00148] The words "example" or "exemplary" are used herein to mean serving as an example, instance or illustration. Any aspect or design described herein as "example' or "exemplary" is not necessarily to be construed as preferred or advantageous over other aspects or designs. Rather, use of the words "example" or "exemplary" is intended to present concepts in a concrete fashion. As used in this application, the term "or" is intended to mean an inclusive "or" rather than an exclusive "or." That is, unless specified otherwise, or clear from context, "X includes A or B" is intended to mean any of the natural inclusive permutations. That is, if X includes A; X includes B; or X includes both A and B, then "X includes A or B" is satisfied

under any of the foregoing instances. In addition, the articles “a” and “an” as used in this application and the appended claims should generally be construed to mean “one or more” unless specified otherwise or clear from context to be directed to a singular form. Moreover, use of the term “an implementation” or “one implementation” or “an implementation” or “one implementation” throughout is not intended to mean the same implementation or implementation unless described as such. Also, the terms "first," "second," "third," "fourth," etc. as used herein are meant as labels to distinguish among different elements and may not necessarily have an ordinal meaning according to their numerical designation.

Claims

What is claimed is:

1. A processing system, comprising:

a control register; and

a processing core, communicatively coupled to the control register;

wherein the processing core is to:

receive a node allocation request specifying an expected running time of an application and a requested number of nodes of a cluster of nodes;

determine, in view of the node allocation request and a current load on the plurality of nodes, an actual number of nodes to be allocated to the application, wherein the actual number of nodes to be allocated to the application optimizes a cluster load criterion; and

notify, using the control register, the application of the actual number of nodes.

2. The processing system of claim 1, wherein notifying the application of the actual number of nodes comprises:

storing the actual number of nodes in a memory data structure that is employed to store a configuration of nodes associated with the application; and

storing an address of the memory data structure in the control register.

3. The processing system of claim 2, wherein storing the address of the memory data structure in the control register comprises obtaining a lock of the control register.

4. The processing system of claim 1, wherein the processing core is further to:

cause the application to be executed using the actual number of nodes.

5. The processing system of claim 1, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any currently running application within a certain time period.

6. The processing system of claim 1, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any running application within a certain time period, multiplied by the time period.

7. The processing system of claim 1, wherein determining the actual number of nodes further comprises:

generating a plurality of combinations of applications to be scheduled;

determining a value of the cluster load criterion for each generated combination; and

selecting a combination that is associated with an optimal value of the cluster load criterion.

8. The processing system of any of claims 1-7, wherein the processing system is implemented as a System-on-Chip (SoC).

9. A method, comprising:

receiving, by a processing device, a node allocation request specifying an expected running time of an application and a requested number of nodes of a cluster of nodes;

determining, in view of the node allocation request and a current load on the plurality of nodes, an actual number of nodes to be allocated to the application, wherein the actual number of nodes to be allocated to the application optimizes a cluster load criterion; and

notifying, using a control register of the processing system, the application of the actual number of nodes.

10. The method of claim 9, wherein notifying the application of the actual number of nodes comprises:

storing the actual number of nodes in a memory data structure that is employed to store a configuration of nodes associated with the application; and

storing an address of the memory data structure in the control register.

11. The method of claim 10, wherein storing the address of the memory data structure in the control register comprises obtaining a lock of the control register.

12. The method of claim 9, further comprising:

causing the application to be executed using the actual number of nodes.

13. The method of claim 9, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any currently running application within a certain time period.

14. The method of claim 9, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any running application within a certain time period, multiplied by the time period.

15. The method of claim 9, wherein determining the actual number of nodes further comprises:

generating a plurality of combinations of applications to be scheduled;

determining a value of the cluster load criterion for each generated combination; and

selecting a combination that is associated with an optimal value of the cluster load criterion.

16. An apparatus comprising:

a memory; and

a processing device coupled to the memory, the processing device to perform the method of any of claims 9-15.

17. A non-transitory computer-readable storage medium comprising executable instructions that, when executed by a processing system, cause the processing system to perform operations comprising:

receiving a node allocation request specifying an expected running time of an application and a requested number of nodes of a cluster of nodes;

determining, in view of the node allocation request and a current load on the plurality of nodes, an actual number of nodes to be allocated to the application, wherein the actual number of nodes to be allocated to the application optimizes a cluster load criterion; and

notifying, using a control register of the processing system, the application of the actual number of nodes.

18. The non-transitory computer-readable storage medium of claim 17, wherein notifying the application of the actual number of nodes comprises:

storing the actual number of nodes in a memory data structure that is employed to store a configuration of nodes associated with the application; and

storing an address of the memory data structure in the control register.

19. The non-transitory computer-readable storage medium of claim 18, wherein storing the address of the memory data structure in the control register comprises obtaining a lock of the control register.

20. The non-transitory computer-readable storage medium of claim 17, further comprising executable instructions causing the processing system to:

cause the application to be executed using the actual number of nodes.

21. The non-transitory computer-readable storage medium of claim 17, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any currently running application within a certain time period.

22. The non-transitory computer-readable storage medium of claim 17, wherein the cluster load criterion reflects a number of nodes that remain unassigned to any running application within a certain time period, multiplied by the time period.

23. The non-transitory computer-readable storage medium of claim 17, wherein determining the actual number of nodes further comprises:

generating a plurality of combinations of applications to be scheduled;

determining a value of the cluster load criterion for each generated combination; and

selecting a combination that is associated with an optimal value of the cluster load criterion.

24. A high-performance computing system, comprising:

a control node; and

a compute cluster comprising a plurality of compute nodes;

wherein the control node is to:

receive a compute node allocation request specifying an expected running time of an application and a requested number of compute nodes of the compute cluster;

determine, in view of the compute node allocation request and a current load on the plurality of compute nodes, an actual number of compute nodes to be allocated to the application, wherein the actual number of compute nodes to be allocated to the application optimizes a cluster load criterion; and

notify the application of the actual number of compute nodes.

25. The high-performance computing system of claim 24, wherein determining the actual number of nodes further comprises:

generating a plurality of combinations of applications to be scheduled;

determining a value of the cluster load criterion for each generated combination; and

selecting a combination that is associated with an optimal value of the cluster load criterion.

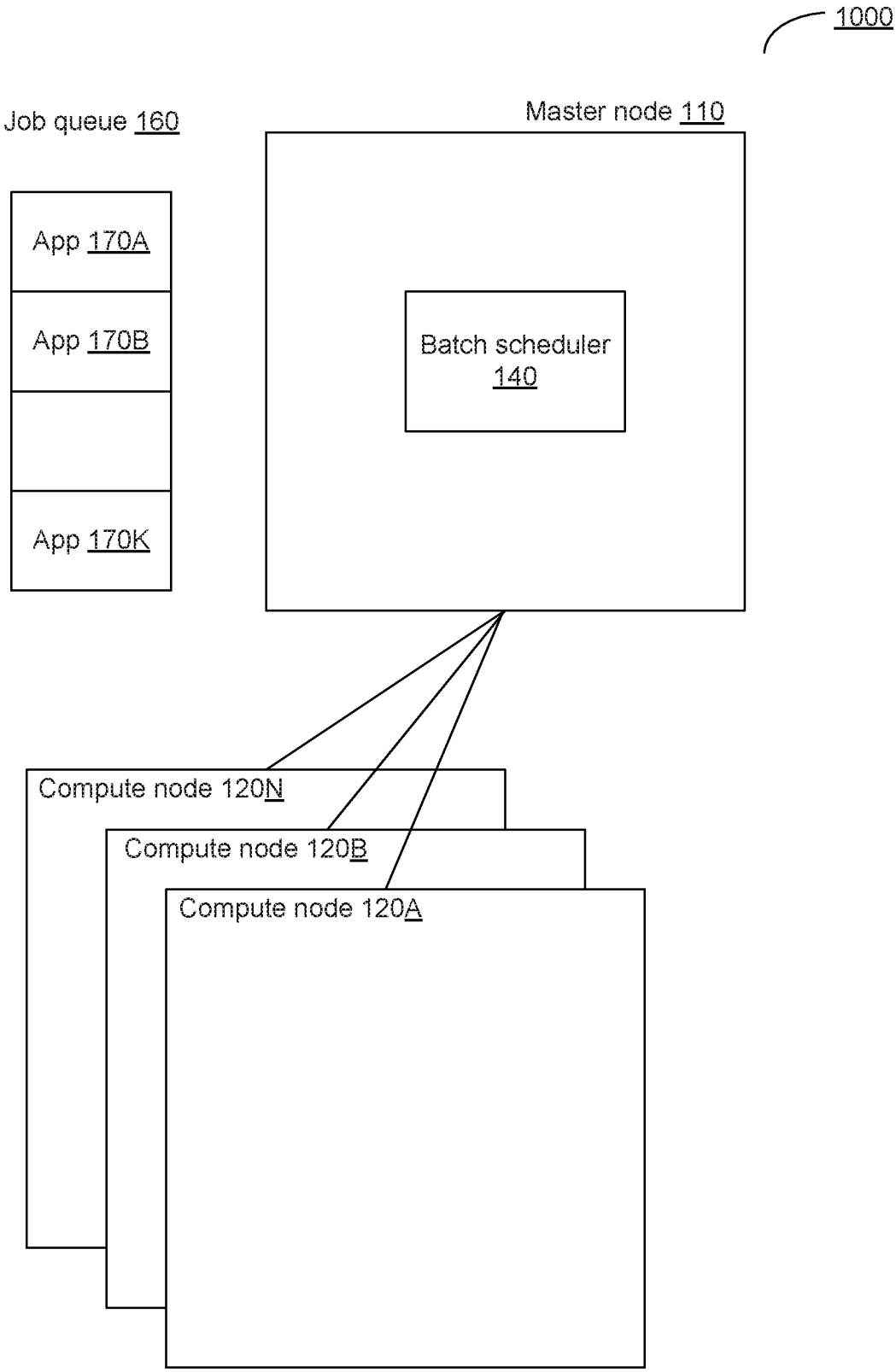

1000
Job queue 160
Master node 110
App 170A
App 170B
App 170K
Batch scheduler 140
Compute node 120N
Compute node 120B
Compute node 120A

FIGURE 1

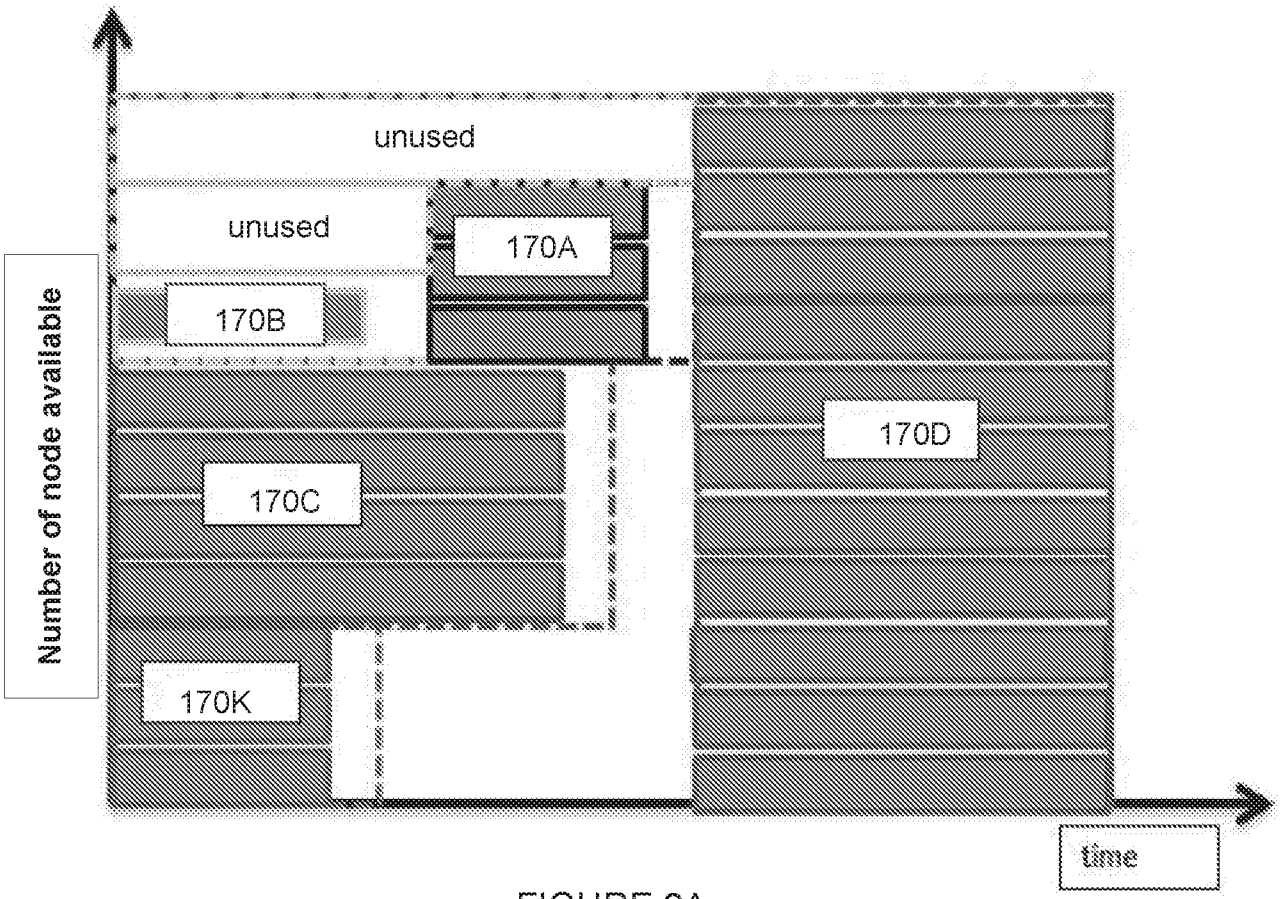


FIGURE 2A

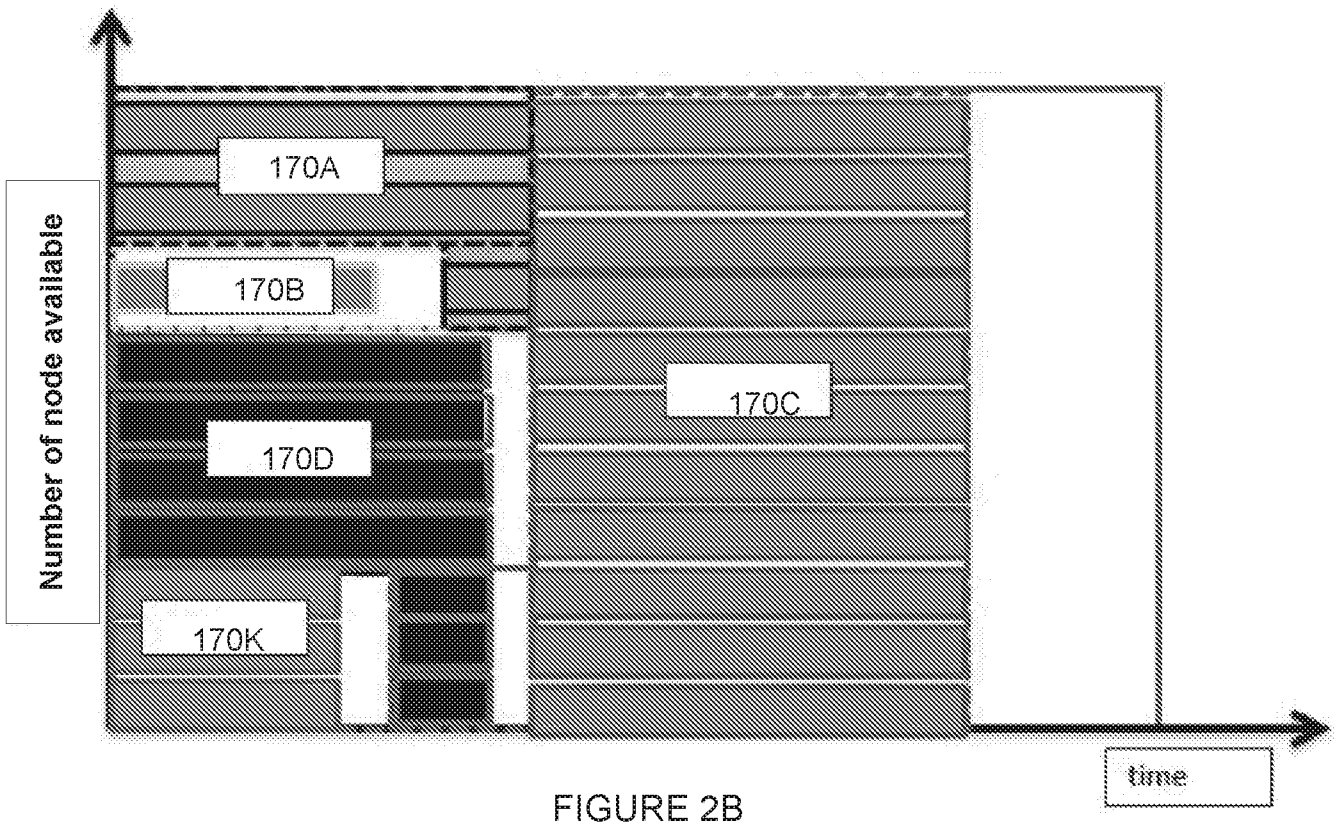


FIGURE 2B

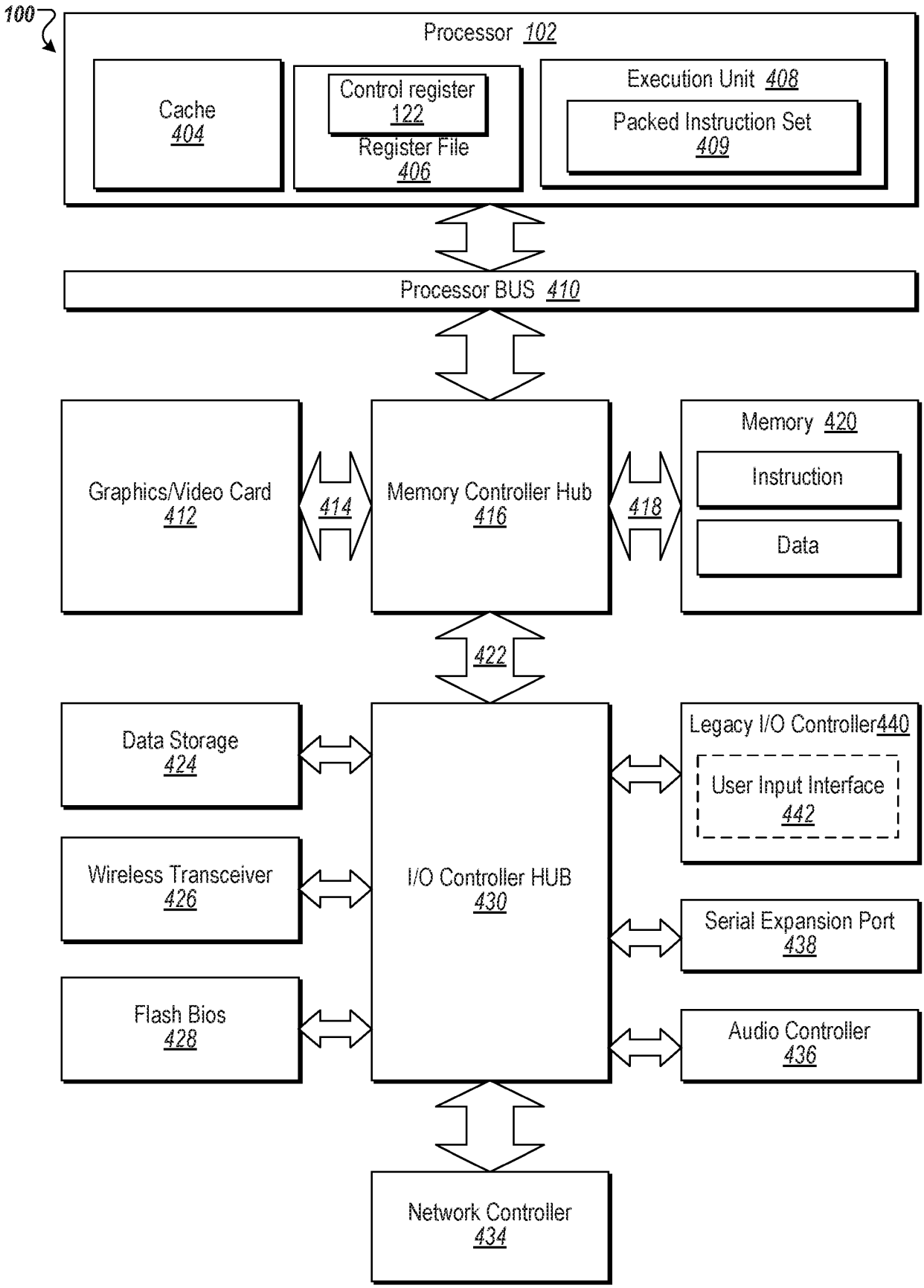

100
Processor 102
Cache 404
Control register 122
Register File 406
Execution Unit 408
Packed Instruction Set 409
Processor BUS 410
Graphics/Video Card 412
414
Memory Controller Hub 416
418
Memory 420
Instruction
Data
422
Data Storage 424
Wireless Transceiver 426
Flash Bios 428
I/O Controller HUB 430
Legacy I/O Controller440
User Input Interface 442
Serial Expansion Port 438
Audio Controller 436
Network Controller 434

FIGURE 3

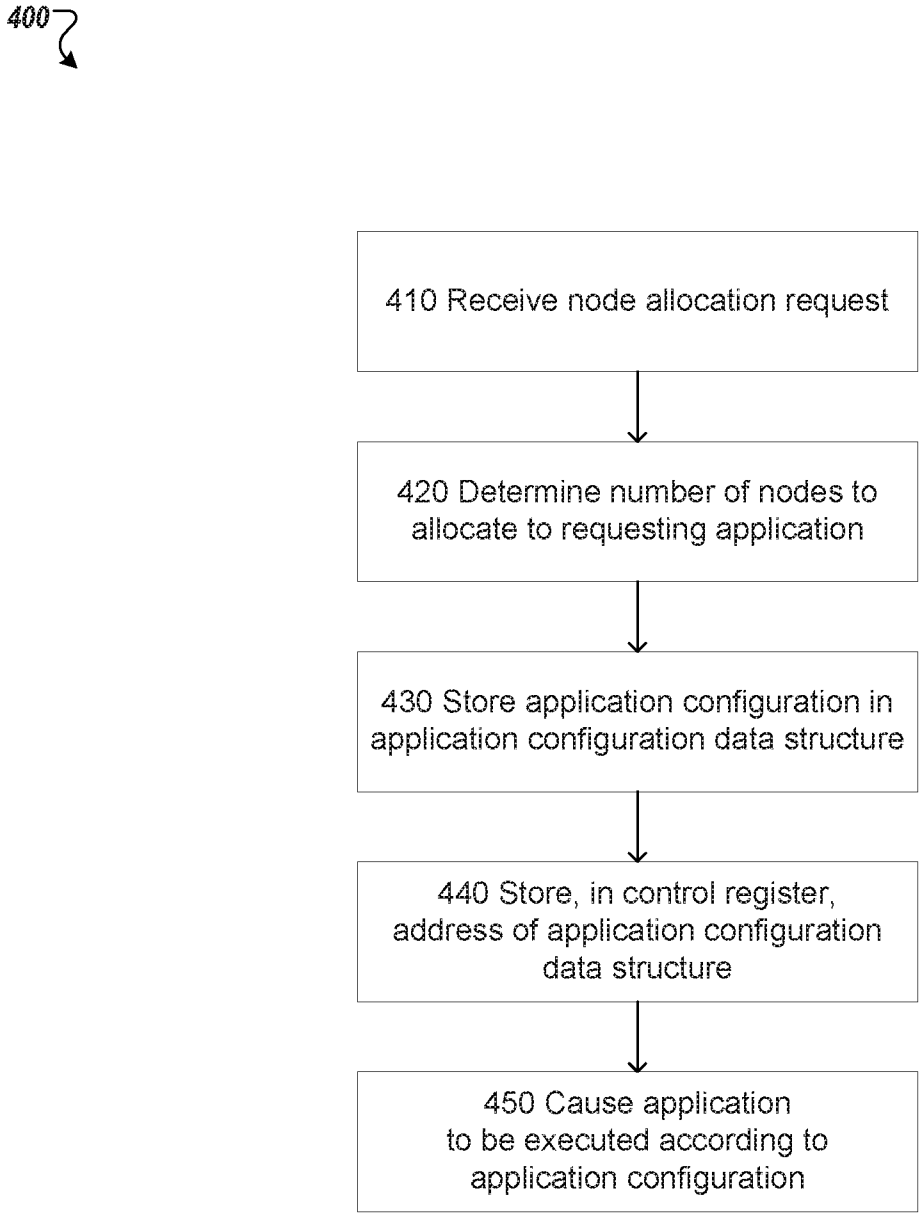

400
410 Receive node allocation request
420 Determine number of nodes to allocate to requesting application
430 Store application configuration in application configuration data structure
440 Store, in control register, address of application configuration data structure
450 Cause application to be executed according to application configuration

FIGURE 4

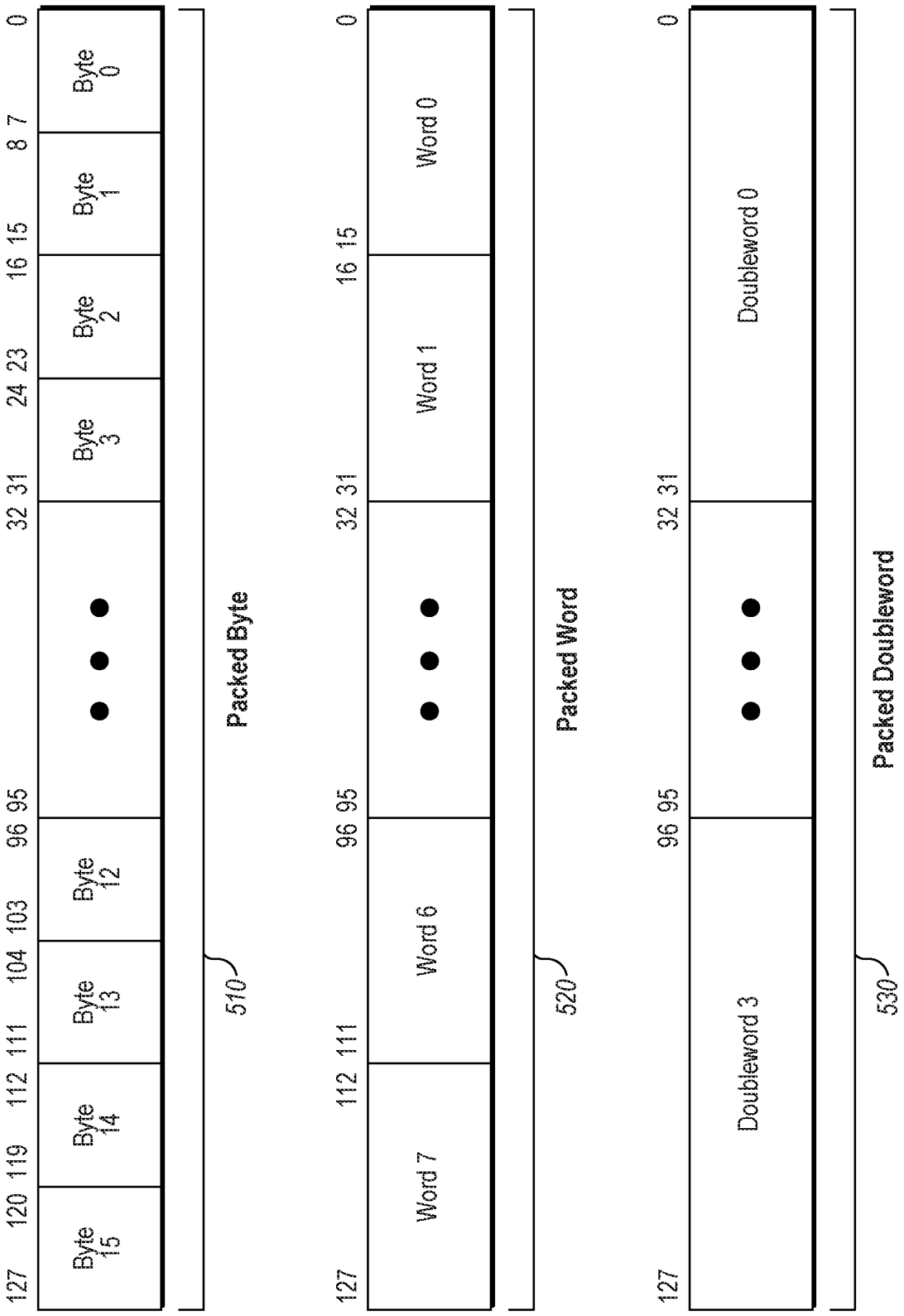

127 120 119 112 111 104 103 96 95 32 31 24 23 16 15 8 7 0
Byte 15
Byte 14
Byte 13
Byte 12
Byte 3
Byte 2
Byte 1
Byte 0
Packed Byte
510
127 112 111 96 95 32 31 16 15 0
Word 7
Word 6
Word 1
Word 0
Packed Word
520
127 96 95 32 31 0
Doubleword 3
Doubleword 0
Packed Doubleword
530

FIGURE 5

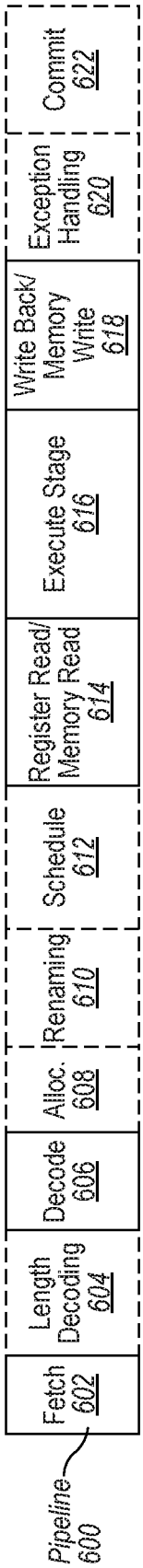


FIGURE 6A

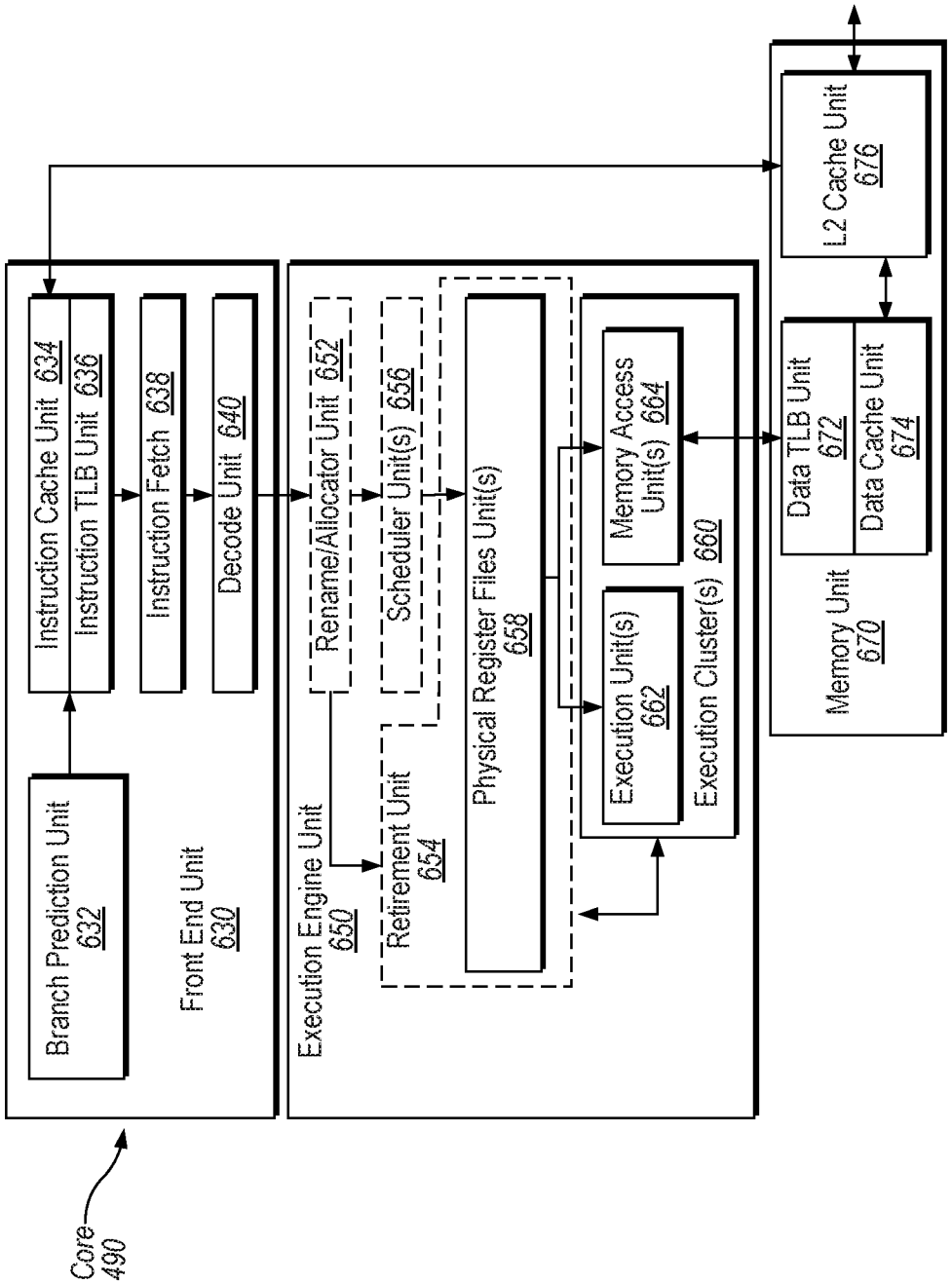


FIGURE 6B

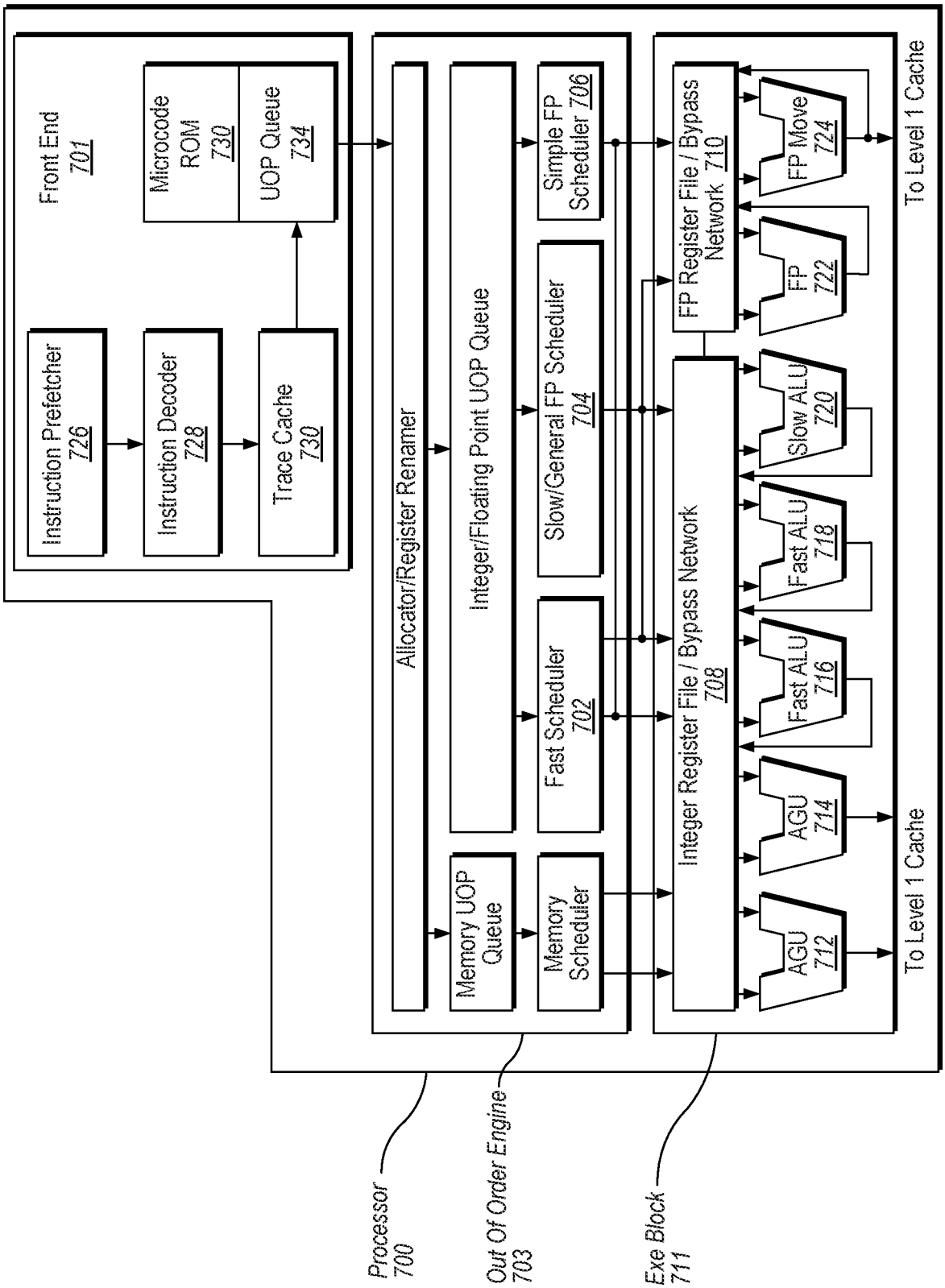

Front End 701
Instruction Prefetcher 726
Instruction Decoder 728
Trace Cache 730
Microcode ROM 730
UOP Queue 734
Allocator/Register Renamer
Integer/Floating Point UOP Queue
Simple FP Scheduler 706
Slow/General FP Scheduler 704
Fast Scheduler 702
Memory UOP Queue
Memory Scheduler
FP Register File / Bypass Network 710
Integer Register File / Bypass Network 708
FP Move 724
FP 722
Slow ALU 720
Fast ALU 718
Fast ALU 716
AGU 714
AGU 712
To Level 1 Cache
To Level 1 Cache
Processor 700
Out Of Order Engine 703
Exe Block 711

FIGURE 7

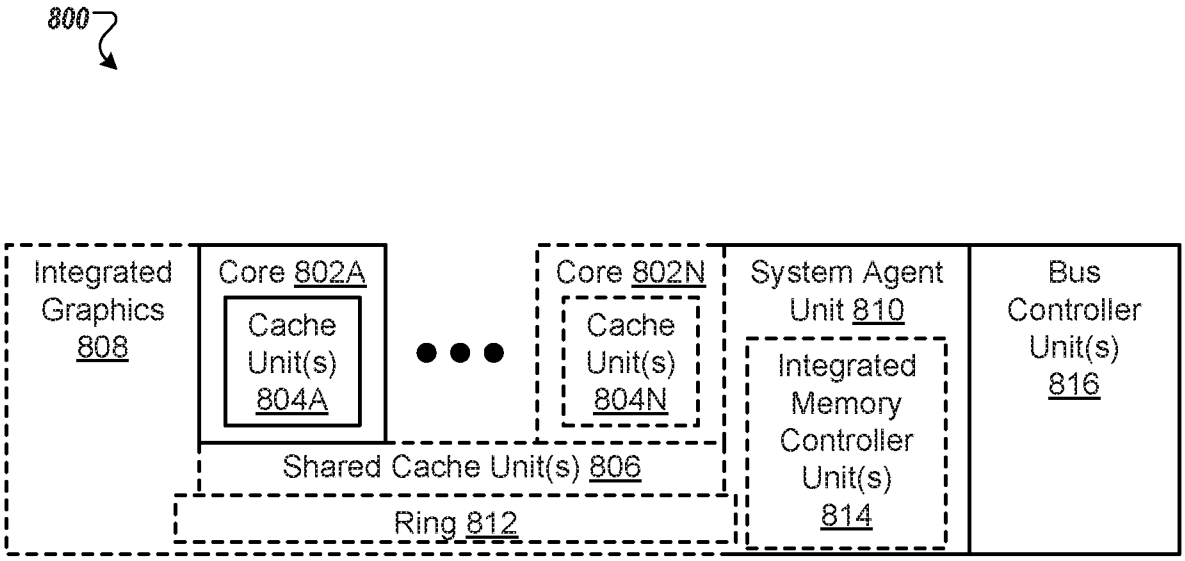

800
Integrated Graphics 808
Core 802A
Cache Unit(s) 804A
Core 802N
Cache Unit(s) 804N
Shared Cache Unit(s) 806
Ring 812
System Agent Unit 810
Integrated Memory Controller Unit(s) 814
Bus Controller Unit(s) 816

FIGURE 8

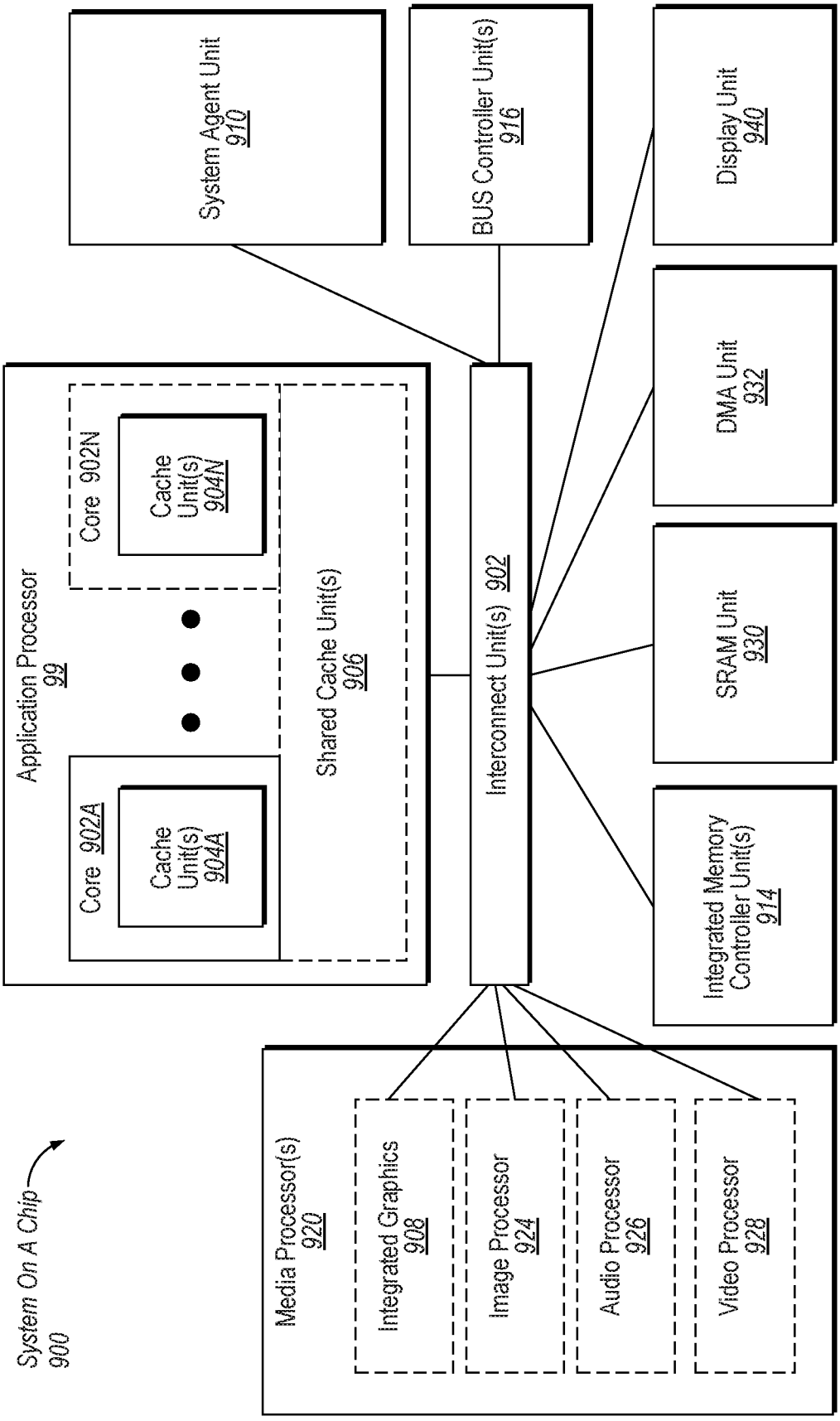

System On A Chip
900
Application Processor
99
Core 902A
Cache Unit(s) 904A
Core 902N
Cache Unit(s) 904N
Shared Cache Unit(s) 906
Interconnect Unit(s) 902
System Agent Unit 910
BUS Controller Unit(s) 916
Display Unit 940
DMA Unit 932
SRAM Unit 930
Integrated Memory Controller Unit(s) 914
Media Processor(s) 920
Integrated Graphics 908
Image Processor 924
Audio Processor 926
Video Processor 928

FIGURE 9

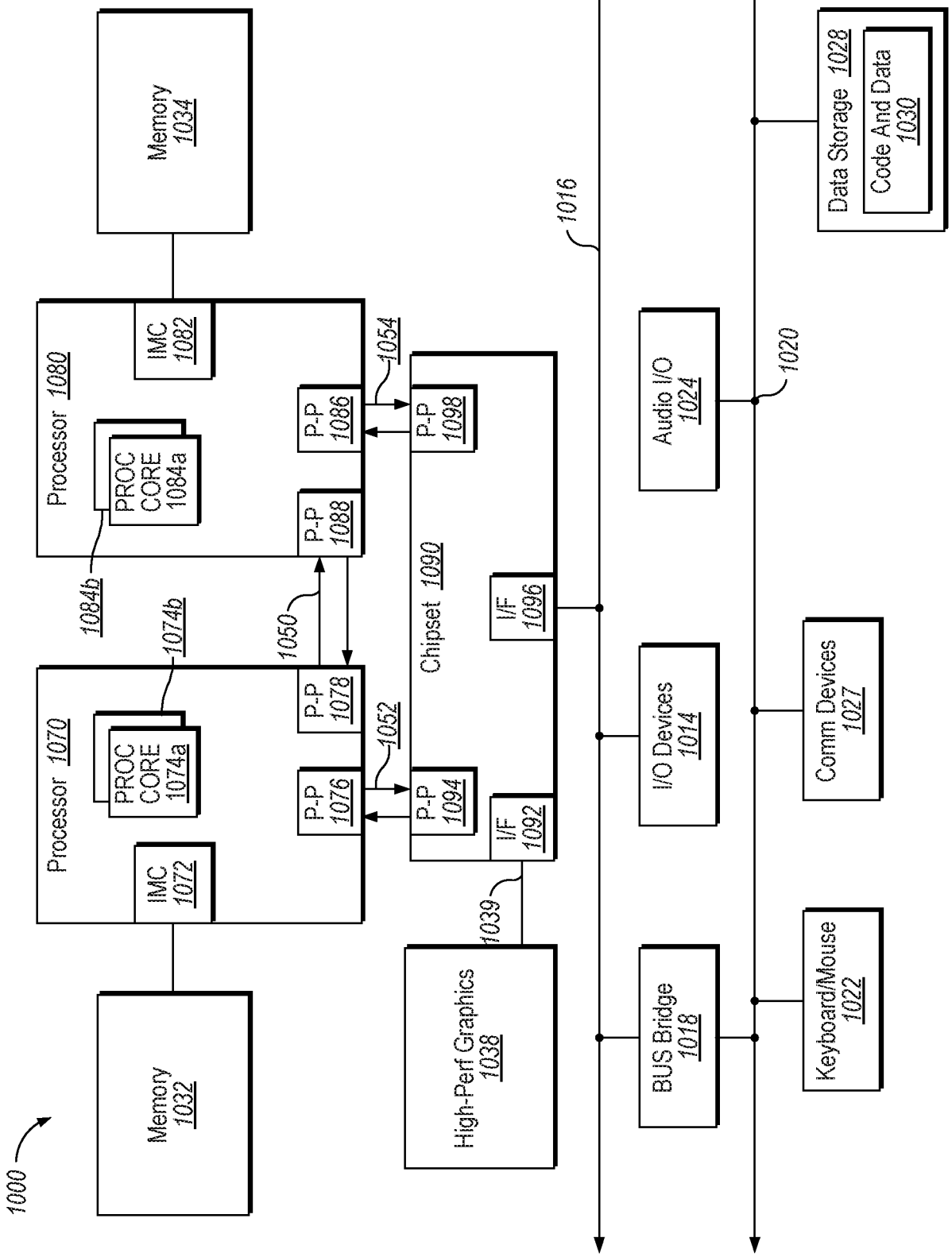

1000
Memory 1032
Memory 1034
Processor 1070
IMC 1072
PROC CORE 1074a
1074b
Processor 1080
IMC 1082
PROC CORE 1084a
1084b
P-P 1076
P-P 1078
P-P 1088
P-P 1086
1050
1052
1054
Chipset 1090
P-P 1094
P-P 1098
I/F 1092
I/F 1096
1039
High-Perf Graphics 1038
1016
BUS Bridge 1018
I/O Devices 1014
Audio I/O 1024
1020
Keyboard/Mouse 1022
Comm Devices 1027
Data Storage 1028
Code And Data 1030

FIGURE 10

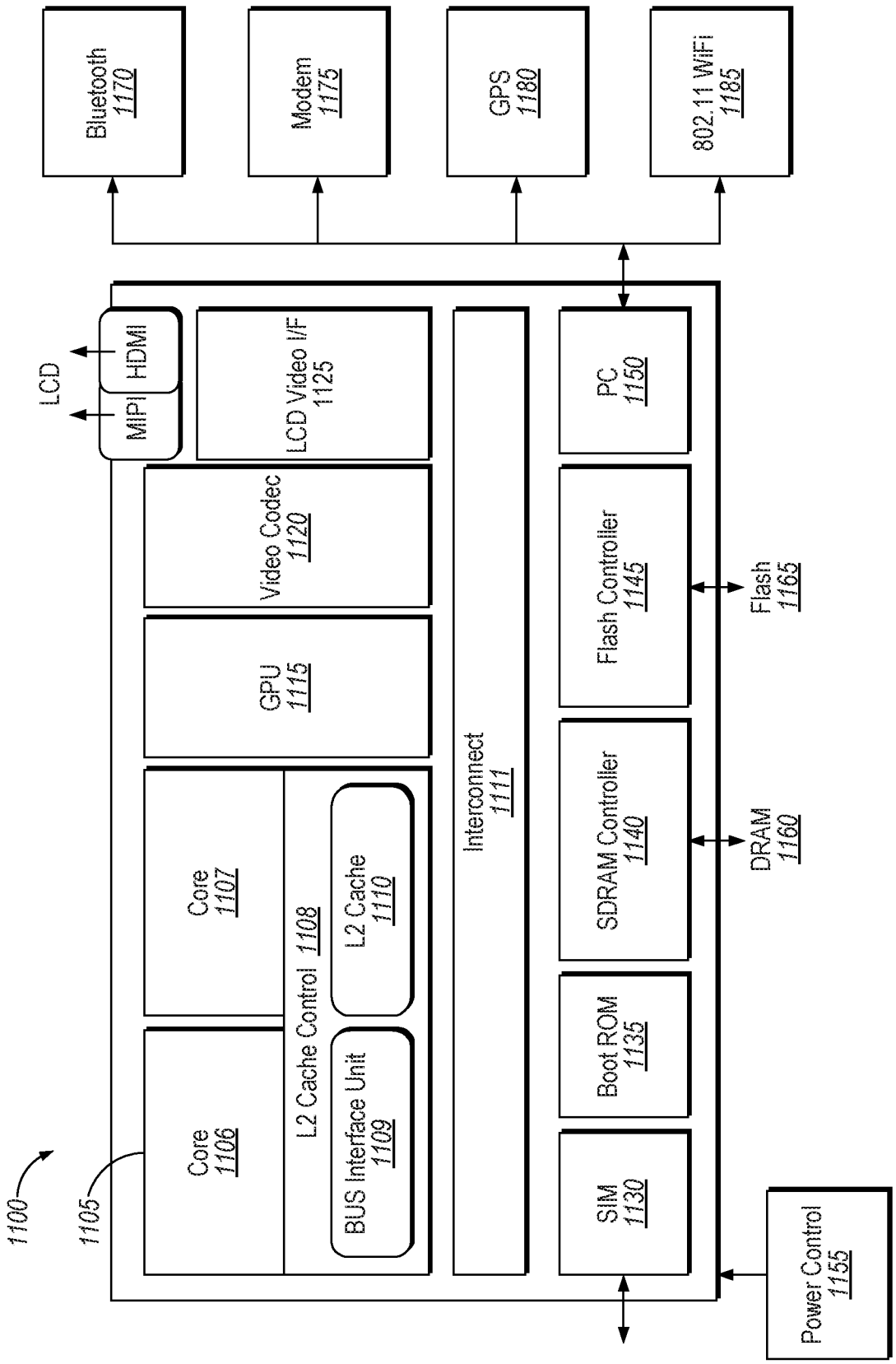

Bluetooth 1170
Modem 1175
GPS 1180
802.11 WiFi 1185
LCD
MIPI
HDMI
LCD Video I/F 1125
PC 1150
Video Codec 1120
Flash Controller 1145
Flash 1165
GPU 1115
Interconnect 1111
Core 1107
L2 Cache Control 1108
L2 Cache 1110
SDRAM Controller 1140
DRAM 1160
Core 1106
BUS Interface Unit 1109
Boot ROM 1135
SIM 1130
Power Control 1155
1100
1105

FIGURE 11

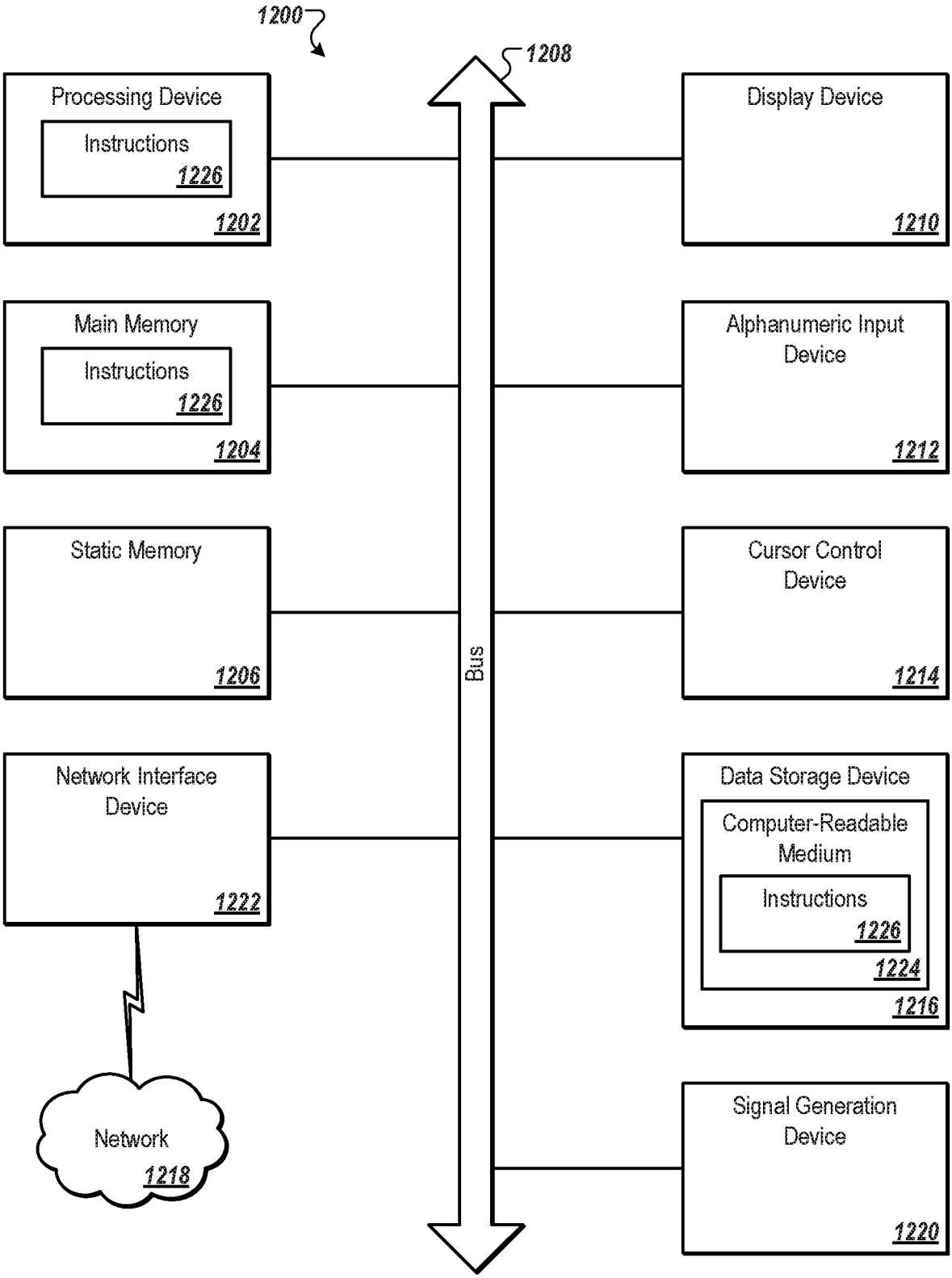

1200
1208
Processing Device
Instructions
1226
1202
Main Memory
Instructions
1226
1204
Static Memory
1206
Network Interface Device
1222
Network
1218
Bus
Display Device
1210
Alphanumeric Input Device
1212
Cursor Control Device
1214
Data Storage Device
Computer-Readable Medium
Instructions
1226
1224
1216
Signal Generation Device
1220

FIGURE 12

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2016/063170

A. CLASSIFICATION OF SUBJECT MATTER

G06F 9/50(2006.01)i

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
G06F 9/50; G06F 9/48; G06F 15/16

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched
Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)
eKOMPASS(KIPO internal) & keywords: allocation request, number of nodes, load criteria, expected running time, cluster, application

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2015-0135185 A1 (AMAZON TECHNOLOGIES, INC.) 14 May 2015 See paragraphs [0013]-[0087], [0103]-[0108]; claim 29; and figures 1A-3.	1-25
A	US 2009-0193427 A1 (XIAOYAN PU et al.) 30 July 2009 See paragraphs [0017]-[0048]; claims 1-7; and figures 2-3.	1-25
A	US 2008-0229320 A1 (HARUYASU UEDA) 18 September 2008 See paragraphs [0047]-[0059]; claim 1; and figures 1-4.	1-25
A	US 2014-0040911 A1 (SEARCHLITE ADVANCES LLC.) 06 February 2014 See paragraphs [0019]-[0055]; claims 49, 65; and figures 1-3.	1-25
A	US 2013-0262556 A1 (MINGQUANG XU et al.) 03 October 2013 See paragraphs [0021]-[0041]; claim 1; and figures 1-3.	1-25

☐ Further documents are listed in the continuation of Box C.

☒ See patent family annex.

* Special categories of cited documents:
"A" document defining the general state of the art which is not considered to be of particular relevance
"E" earlier application or patent but published on or after the international filing date
"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
"O" document referring to an oral disclosure, use, exhibition or other means
"P" document published prior to the international filing date but later than the priority date claimed
"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents,such combination being obvious to a person skilled in the art
"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
28 February 2017 (28.02.2017)	**02 March 2017 (02.03.2017)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon, 35208, Republic of Korea
Facsimile No. +82-42-481-8578

Authorized officer
CHIN, Sang Bum
Telephone No. +82-42-481-8398

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.
PCT/US2016/063170

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2015-0135185 A1	14/05/2015	US 8719415 B1	06/05/2014
		US 8966030 B1	24/02/2015
		US 9280390 B2	08/03/2016
US 2009-0193427 A1	30/07/2009	US 8281012 B2	02/10/2012
US 2008-0229320 A1	18/09/2008	JP 2008-226181 A	25/09/2008
US 2014-0040911 A1	06/02/2014	US 2016-0085592 A1	24/03/2016
		US 7065764 B1	20/06/2006
		US 7937705 B1	03/05/2011
		US 8544017 B1	24/09/2013
		US 9122529 B2	01/09/2015
US 2013-0262556 A1	03/10/2013	None	