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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
4 February 2016 (04.02.2016)

WIPO | PCT

(10) International Publication Number
WO 2016/018400 A1

(51) International Patent Classification:
G06F 17/00 (2006.01)

(21) International Application Number:
PCT/US2014/049264

(22) International Filing Date:
31 July 2014 (31.07.2014)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **HEWLETT-PACKARD DEVELOPMENT COMPANY, L.P.** [US/US]; 11445 Compaq Center Drive W., Houston, Texas 77070 (US).

(72) Inventors: **HUANG, Muhuan**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **KEETON, Kimberly**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **MORREY, III, Charles B.**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US). **LIM, Kevin T.**; 1501 Page Mill Road, Palo Alto, California 94304-1100 (US).

(74) Agents: **ORTEGA, Arthur S.** et al.; Hewlett-Packard Company, Intellectual Property Administration, Mail Stop 35, 3404 E. Harmony Road, Fort Collins, Colorado 80528 (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, 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, KZ, LA, LC, LK, LR, LS, LT, 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, 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, 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).

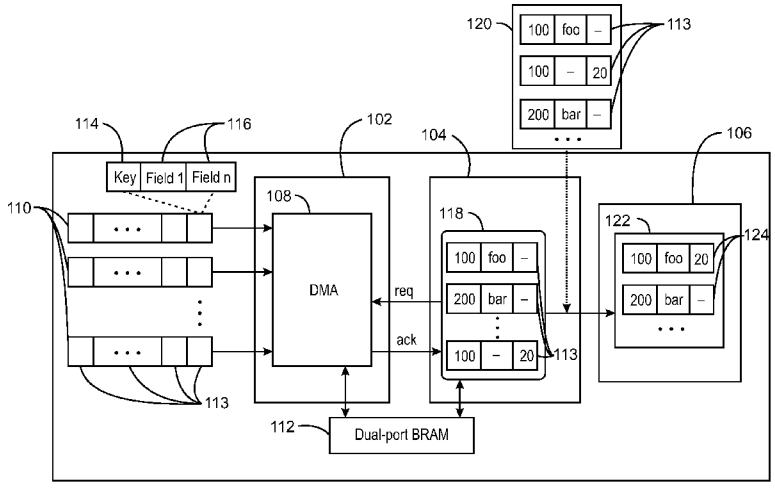
Declarations under Rule 4.17:

— *as to the identity of the inventor (Rule 4.17(i))*

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

[Continued on next page]

(54) Title: DATA MERGE PROCESSING



100
FIG. 1

(57) Abstract: Techniques are described in which updates are received. The updates are ordered by priority value via a priority queue. A higher priority update is received from the priority queue. The higher priority update is merged with a corresponding database row.

WO 2016/018400 A1

Published:

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

DATA MERGE PROCESSING

BACKGROUND

[0001] In computing, there are several examples of processes that merge large amounts of data. Data may also be prioritized using values that indicate the relative priority of individual data.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] Various features of the techniques of the present application will become apparent from the following description of examples, given by way of example only, which is made with reference to the accompanying drawings, of which:

[0003] Fig. 1 is a block diagram of an example merge stage;

[0004] Fig. 2 is a process flow diagram illustrating an example method for prioritizing data to be merged;

[0005] Fig. 3A is a diagram of an example priority processor populated with initial values;

[0006] Fig. 3B is a diagram of an example priority processor fully populated with an ordered array of individual updates with priority values displayed;

[0007] Fig. 3C is a diagram of an example priority processor receiving a new individual update;

[0008] Fig. 3D is a diagram of an example priority processor having processed a new individual update;

[0009] Fig. 3E is a diagram of an example priority processor receiving another individual update;

[0010] Fig. 3F is a diagram of an example priority processor having processed the individual updates;

[0011] Fig. 4 is a process flow diagram illustrating an example method for merging SCUs;

[0012] Fig. 5 is block diagram of an example computing device to merge data; and

[0013] Fig. 6 is a drawing of an example machine-readable storage medium that can be used to merge data.

DETAILED DESCRIPTION

[0014] There are processes in computing environments that may benefit from merging data efficiently. Data may be prioritized using values that indicate the relative priority of individual data. For example, in databases, data ordering may be based on the primary keys in an index. For example, the primary key values may be used to retrieve or update information in a specified order.

[0015] Improved merging of data flows may be achieved using priority queues. As used herein, priority queues refer to data structures that are used to prioritize individual updates in batch updates to data entries. The priority queues described herein may maintain a partially ordered internal structure, but allow for constant time identification of a priority element. Thus, priority queues may be used to provide a fast ordering where a priority element is the item of interest. For example, a priority element in a priority queue may be an individual update with an entry having a higher primary index number than other individual updates in the queue. In addition, a specialized processor such as a field-programmable gate array (FPGA) may be used to achieve parallel processing of elements to be merged.

[0016] Some current designs process pipeline stages on a standard processor. As used herein, pipelined refers to a set of elements being connected in series, where the output of one element is the input of the next element, and multiple operations may occur in parallel along the series. By executing on a standard processor, the pipeline stages have the limitations of using a general purpose processor. For example, the architecture of the processor may not be well suited for the access patterns nor enable as much parallelism.

[0017] Described herein are techniques relating to merging data using a specialized priority processor and merging processor in a computing device. In some examples, a priority processor of the computing device may prioritize data in a pipelined array associated with a priority queue as described in Figs. 3A-3F below. The prioritized data can then be efficiently merged using a merging processor. An example merge stage is shown in Fig. 1 and an example of such computing device is shown in Fig. 5.

[0018] Fig. 1 is a detailed block diagram of an example merge stage according to implementations described herein. The configuration of the example merge stage is generally referred to by the reference number 100. The merge stage includes three primary components, including a fetch stage 102, a priority queue 104, and a data

merge stage 106. The data fetch stage 102 includes a Direct Memory Access (DMA) engine 108 that reads data such as individual updates 113 of self-consistent updates (SCUs) 110 from local memory into block random access memory (BRAM) buffers 112. As used herein, an SCU refers to a batch of updates that are received by a client node and/or a pointer to the batch of updates. SCUs include a set of changes that are applied atomically to tables in a database. In some examples, the SCUs are stored durably onto a disk when they are first uploaded. The SCUs are said to be self-consistent to the extent that each SCU is applied consistently such that all underlying tables in a system are consistent after the changes are applied. Moreover, the application of an SCU is isolated from other SCUs. Each SCU 110 includes individual updates 113 that can each include a key 114 and one or more related fields 116. In some examples, one or more fields 116 may not have an update to be applied as indicated by dashes in updates 113. The priority queue 104 includes a pipelined array 118 of individual updates 113 to be prioritized and a prioritized array 120 of individual updates 113. The merge stage 106 includes an array 122 of merged individual updates 124.

[0019] In some examples, the fetch stage 102 may begin with the DMA engine reading data from a local memory to BRAM buffers. In some examples, the BRAM buffers may be an on-FPGA memory. In some examples, the DMA may receive a request for a particular data in the form of a request for acknowledgment. For example, a priority processor can request an individual update from a self-consistent update (SCU) from the DMA. In some examples, the updates can be batched based on a specific time interval. For example, the updates can be batched once every several seconds. In some examples, updates can be batched based on a specific number of updates. For example, the updates can be batched every 100 updates. In some examples, the DMA can return an acknowledgment to the priority queue and retrieve the individual update from an SCU. In some examples, the BRAM buffers provide temporary storage for a plurality of individual updates 113 from each SCU 110. The BRAM thus enables priority queue operation to occur while overlapping the transfer from the local memory.

[0020] In some examples, the priority queue 104 may be a high speed parallel implementation that utilizes a pipelined array. A pipelined array allows replace operations to occur on the priority queue 104 every clock cycle. In some examples, the pipelined array may be processed by a priority processor, such as an FPGA or

Application Specific Integrated Circuit (ASIC). The FPGAs enable the sorting of the array implementing the priority queue to occur in parallel with the replace operation. This may result in a 10x improvement over standard software implementations. Standard software implementations take logarithmic time between replace operations, whereas the time spent between replace operations in the present implementations is fairly constant without regard to the number of operations concurrently executed.

[0021] In some examples, the data merge stage 106 takes each prioritized individual update 113 provided by the priority queue 104 and merges the data of each prioritized individual update 113 with the current version of the specific row. Each update potentially includes updates to multiple fields within the row. For each field 116, a timestamp is be compared to each field's timestamp in the current version of the row to see if the prioritized individual update 113 has more recent data. If the prioritized individual update 113 has more recent data, then the current version's data is changed to match the update 113. In some examples, a merge processor implemented using an FPGA is able to compare timestamps across multiple fields in parallel, allowing a very fast merging of data. Traditional software approaches often examine each field sequentially. In some examples, the final merged updates 124 of merged array 122 are written out to local memory once the latest higher priority update of the priority queue 104 indicates that there are no more updates to that row. For example, a new value for the primary key may be returned by the priority queue indicating an update to another row.

[0022] Fig. 2 is a process flow diagram illustrating an example method for prioritizing data to be merged. The method of Fig. 2 is generally referred to by the reference number 200.

[0023] At block 202, a processor initializes a priority queue 104. As used herein, higher numbers represent higher priority and lower numbers represent lower priority. In some examples, because the replace operation is used, initial values are used to initialize the priority queue. In some examples, the processor may populate a priority queue 104 with values indicating infinity. In some examples, the processor may use lower values to indicate priority and may populate the priority queue 104 with values indicating negative infinity. In both cases, the infinity values serve as initial placeholders in the priority queue 104 that may be replaced by individual updates 113 using the replace operation.

[0024] At block 204, the priority queue 104 receives an individual update 113 from the SCUs 110. For example, the update 113 may correspond to one of a plurality of database entries to be updated. The update 113 may include the data to be updated or a pointer to the update 113. For example, update 113 may refer to one of a batch of updates in an SCU or a pointer that identifies an individual update in the SCU. Each individual update also includes a priority value associated with the individual update. The priority value may be assigned by the processor. In some examples, the priority value may be a key value such as primary key 114. For example, the key value may be used to determine a relative position in a database entry to be updated. In some examples, related database entries to be updated may be later merged together using the key value and updated in a more efficient manner.

[0025] At block 206, the priority processor identifies a higher priority update 113 than all other updates 113 in the priority queue and replaces the higher priority update 113 at the root position in a priority queue with a new update 113 from SCUs 110. A root position in a priority queue, as used herein, refers to a level in a priority queue that receives new updates 113 from the SCUs 110 and also contains higher priority updates 113. After identifying the higher priority update 113, the priority processor may send the identified update 113 to the other processor to indicate the corresponding update 113 to be updated. In some examples, the priority processor may send the identified higher priority updates 113 of prioritized array 120 to a memory for merging in the merge stage 106. In some examples, the root level of the priority queue may contain the higher priority update 113 after a complete clock cycle. In some examples, the root level may contain the new update 113 from SCUs 110 after a complete clock cycle. Thus, by replacing the update 113 at the root position of the priority queue 104 with a new update 113 from the SCUs 110, the priority processor may identify an update 113 with a higher priority than all other updates 113 in the array 118.

[0026] At block 208, the priority processor swaps even-level updates 113 with consecutively higher odd-level updates 113 based on a comparison of priority values associated with the updates 113. For example, the priority values may be key values 114 associated with each update 113. In some examples, given six levels 0-5, the updates 113 in levels 0 and 1 may be swapped, the updates 113 in levels 2 and 3 may be swapped, and the updates 113 in levels 4 and 5 may be swapped. In

some examples, the updates 113 of two levels are swapped based on their priority values. For example, the updates 113 may be swapped when the higher level update 113 has a higher priority value. For example, in a priority queue, the higher priority updates 113 will be sorted to lower levels. Thus, in the priority queue, if an update 113 in level 0 has a priority value of 5 and an update 113 in level 1 has a priority value of 2, then the update 113 in level 0 will not be swapped after being compared with the update 113 in level 1 because they are already sorted correctly. In some examples, the priority processor may simultaneously swap all the pairs of odd/even levels of updates 113 that are to be swapped. In some examples, block 208 may be executed at higher levels concurrently with the execution of block 206 at the lower levels.

[0027] At block 210, the priority processor swaps odd-level updates 113 with consecutively higher even-level updates 113 based on a comparison of priority values associated with the updates 113. For example, the update 113 of level 1 might be swapped with an update 113 of level 2, an update 113 of level 3 might be swapped with an update 113 of level 4, and so on. In some examples, the updates 113 are swapped according to their priority values. For example, an update 113 in level 1 may be swapped with an update 113 in level 2 if the update 113 in level 1 has a lower priority value than the update 113 in level 2. In some examples, block 210 may be executed at higher levels concurrently with the execution of block 206 at the lower levels.

[0028] In some examples, as indicated by diamond 212, if additional updates 113 are received by the priority queue 104, then method 200 may iterate through additional received updates 113 by cycling through blocks 204-210. If no further additional updates 113 are received, then the method proceeds to diamond 214.

[0029] In some examples, as indicated by diamond 214, if no additional updates 113 are received by priority queue 104, then method 200 may proceed to cycle through blocks 208-210 until no further swaps are performed because all the updates 113 are sorted. In some examples, the priority processor may be populated with lower priority values to sort the remaining sorted updates 113 and identify higher priority updates 113 in the priority queue. If all the updates 113 are sorted, then the method ends at block 216.

[0030] It is to be understood that the process diagram of Fig. 2 is not intended to indicate that all of the elements of the method 200 are to be included in every case.

For example, a clock cycle may begin at block 204 and end at block 210. In some examples, a clock cycle may begin at block 208, proceed to block 210, then finish with blocks 204 and 206. Further, any number of additional elements not shown in Fig. 2 may be included in the method 200, depending on the details of the specific implementation.

[0031] Fig. 3A is a diagram of an example priority processor initialized with initial values. The configuration of the example priority processor of Fig. 3A is referred to generally by the reference number 300A. The priority processor 300A includes levels 0-7 that are labeled as levels 302-316, respectively. In the example of 300A, levels 302-316 are populated by the value infinity 318.

[0032] In Fig. 3A, the levels 302-316 are populated by the value infinity 318 because priority is indicated by higher priority values. For example, the priority queue may be arranged to output the priority updates and store the rest of the updates in a descending order from left to right. In some examples, the priority processor 300A may use replace and delete functions, and not insert functions. By using the replace and delete functions on the priority processor 300A in parallel on all the levels of the priority queue, rather than using insert functions, the priority processor 300A may allow an operation following a replacement or removal in O(1), or constant time, instead of O(log n), or logarithmic time. Therefore, the priority processor 300A may efficiently process updates regardless of the total amount of updates to be processed. Furthermore, because a single array is used, the priority processor 300A may use storage space efficiently.

[0033] Fig. 3B is a diagram of an example priority processor fully populated with an ordered array of individual updates 113 with priority values displayed. The configuration of the example priority processor in Fig. 3B is referred to generally by the reference number 300B. Updates 320-334 correspond to levels 302-316 of the priority processor, respectively.

[0034] In the diagram of Fig. 3B, the corresponding priority values of updates 320-334 have replaced the value infinity 318 one clock cycle at a time. In some examples, after eight clock cycles, the order of the updates 320-334 is from higher to lower. The original order of the updates 320-334 does not matter because of the swapping function as discussed at greater length in Fig. 3C. Thus, the priority processor 300B is able to efficiently sort updates regardless of their original order.

[0035] Fig. 3C is a diagram of an example priority processor receiving a new update. The configuration of the priority processor in Fig. 3C is generally referred to by the reference number 300C. In addition, new update 336 is about to replace update 320 as shown by arrow 338. Update 320 is also about to be identified as a higher priority update and sent to output as shown by arrow 340. In some examples, the output may be sent to a processor or memory as discussed further in Figs. 5 and 6 below.

[0036] In the diagram of Fig. 3C, the fully populated priority processor 300C receives a new individual update 336. The new update 336 is received at level 0 302, also referred to herein as the root level 302. In some examples, the priority processor 300C uses the replace function to replace update 320 at root level 302 with new update 336 and output update 320. In some examples, the update 320 may be output 340 to a processor, memory, or storage device. In some examples, the priority processor 300C may then swap consecutive updates using the replace operation as described in Fig. 3D.

[0037] Fig. 3D is a diagram of an example priority processor having processed an individual update 113. The configuration of the priority processor in Fig. 3D is generally referred to by the reference number 300D. A first round of swap and comparisons are indicated by arrows 342 and 344, respectively. A second round of swap and comparisons are indicated by arrows 346 and 348, respectively.

[0038] In the diagram of Fig. 3D, update 336 has shifted two places to the right from root level 302 to level 306. In some examples, the replacement of update 336 with the original update 320 at root level 302 and the shifting of update 336 two levels to the right may be performed by the priority processor 300D within one clock cycle. In some examples, the priority processor 300D may perform two sets of adjacent comparisons and/or swaps. For example, a first set of a swap and comparisons of even-levels with consecutively higher odd-levels indicated by arrows 342 and 344, respectively, results in new update 336 at root level 302 swapping with higher priority update 322 at level 304. Thus, update 322 is then placed into root level 302 and update 336 takes the place of update 322 at level 304. Although comparisons are made as indicated by arrows 344, the priority processor 300D does not perform any swaps because the priority values of these updates indicate that they are already ordered in a descending order of priority. In a second set of swaps and comparisons, as indicated by arrows 346 and 348, update 336 of level 304 is

then swapped with higher priority update 324 of level 306. Thus, update 336 moves up to level 306, and update 324 moves down to level 304, the final resulting order of the updates shown in the example of 300D.

[0039] Fig. 3E is a diagram of an example priority processor receiving another individual update 113. The configuration of the priority processor in Fig. 3E is generally referred to by the reference number 300E. A new update 350 is to replace update 322 as shown by arrow 352. Update 322 is also to be output by the priority processor 300E as shown by arrow 354.

[0040] In the diagram of Fig. 3E, a new update 350 is to be added to the pipeline processor configuration of 300D. As in 300C, the new update 350 is to replace the existing update 322 of root level 302, the existing update 322 to be output by the priority processor 300E as indicated by arrow 354. However, this time two pairs of swaps will simultaneously follow the replacement of root level 302 as described in further detail with reference to Fig. 3F.

[0041] Fig. 3F is a diagram of an example priority processor having processed the individual updates. The configuration of the priority processor in Fig. 3F is generally referred to by the reference number 300F. Two pairs of swaps 342, 346 are indicated by bold dotted arrows, while comparisons 344, 348 are indicated by lightly dotted arrows.

[0042] In Fig. 3F, both new update 350 of 300E and update 336 of 300C have been shifted up two levels to the right. As discussed in 300D, the priority processor 300F executes two consecutive swaps. However, 300F shows two pairs of consecutive swaps. In some examples, more than one update may simultaneously be swapped with a consecutively higher level update. For example, in 300F update 350 of root level 302 was swapped with update 324 of level 304, and update 336 of level 306 was swapped with update 326 of level 308. In some examples, after the even-level updates compared and/or swapped with consecutively higher odd-level updates, then the odd-level updates are compared with the corresponding consecutively higher level even-level updates. For example, in the example of 300F, update 350 at level 304 was compared and swapped with update 326 of level 306, and update 336 of level 308 was compared and swapped with update 328 of level 310. Thus, 300F shows the final positions of the two sets of swaps. As updates 330-334 are still ordered properly with respect to each other, no swaps included updates 330-334. However, in some examples, with two additional clock cycles,

update 336 may eventually reach level 316 as it is an update with a lower priority. Likewise, in some examples, with an additional clock cycle, priority processor 300F may swap update 350 into level 310 and keep it there until a lower priority update is introduced at later clock cycles. Processing a pipelined array on a priority processor 300F such as an FPGA may result in a higher overall performance. For example, an implemented pipelined array priority queue on an FPGA board produced benchmarks indicating about a tenfold speedup over software implementations, and about a threefold speedup over pipelined heap designs.

[0043] It is to be understood that the diagrams of Figs. 3A-3F are not intended to indicate that all of the elements of the configurations 300A-300F are to be included in every case. Further, any number of additional elements not shown in Figs. 3A-3F may be included in the configurations 300A-300F, depending on the details of the specific implementation. For example, in configuration 300F, more than two updates may be swapped at the same time with a consecutively higher level, depending on the priority values of the updates.

[0044] Fig. 4 is a process flow diagram illustrating an example method for merging data. The method of Fig. 4 is generally referred to by the reference number 400.

[0045] At block 402, a priority processor receives individual updates 113. For example, the individual updates 113 may be from SCUs 110 and are to be prioritized and merged with a current database row.

[0046] At block 404, the priority processor orders the updates 113 by priority value via a priority queue. For example, the priority value may be a primary key associated with each individual update. In some examples, the priority processor can order the updates 113 while concurrently sending higher priority updates 113 to the merge processor. In some examples, the DMA can store higher priority updates 113 in the BRAM for the priority processor to access higher priority updates.

[0047] At block 406, a merge processor receives a higher priority update from the priority queue. For example, the higher priority update may have had a primary key that is equal to or higher than all the rest of the updates of the priority queue.

[0048] At block 408, the merge processor merges the higher priority update with a corresponding database row. In some examples, merging the higher priority update may include updating multiple fields of the corresponding database row based on a

timestamp comparison. In some examples, multiple fields are to be updated in parallel.

[0049] At diamond 410, the merge processor checks if a different database row is indicated by a priority value. For example, a primary key can be used to distinguish between members of different rows. If the primary key indicates that the higher priority update corresponds to the same database row as the prior primary key, then the method proceeds back to 408 wherein the merge processor is to receive additional higher priority updates from the priority queue. In some examples, the merge processor may then merge the additional higher priority updates with their corresponding database rows. In some examples, if the primary key indicates that a higher priority update corresponds to a different row based on the primary key, then the method proceeds to 412.

[0050] At block 412, the merge processor sends the merged database row to memory after receiving a higher priority update corresponding to a different database row. In some examples, the merge processor can also store the merged database in a storage device.

[0051] It is to be understood that the process diagram of Fig. 4 is not intended to indicate that all of the elements of the method 400 are to be included in every case. Further, any number of additional elements not shown in Fig. 4 may be included in the method 400, depending on the details of the specific implementation.

[0052] Fig. 5 is a block diagram of an example computing device 502 to update data. The computing device 502 may include a processor 504, memory 506, a machine-readable storage 508, a network interface card (NIC) 510 to connect computing system 102 to network 112, a direct memory access (DMA) engine 514, a priority processor 516, and a merge processor 518.

[0053] In some examples, the processor 504 may be a main processor that is adapted to execute the stored instructions. The processor 504 may be a single core processor, a multi-core processor, a computing cluster, or any number of other configurations. The processor 504 may be implemented as Complex Instruction Set Computer (CISC) or Reduced Instruction Set Computer (RISC) processors, x86 Instruction set compatible processors, ARMv7 Instruction set compatible processors, multi-core, or any other microprocessor or central processing unit (CPU).

[0054] In some examples, the memory device 506 may include random access memory (e.g., SRAM, BRAM, DRAM, zero capacitor RAM, SONOS, eDRAM, EDO

RAM, DDR RAM, RRAM, PRAM, etc.), read only memory (e.g., Mask ROM, PROM, EPROM, EEPROM, etc.), flash memory, or any other suitable memory systems. As described below, in some examples, the memory may receive identified higher priority data from the priority processor 516.

[0055] In some examples, machine-readable storage 508 may be any electronic, magnetic, optical, or other physical storage device that stored executable instructions. Thus, machine-readable storage medium may be, for example, Random Access Memory (RAM), an Electrically-Erasable Programmable Read-Only Memory (EEPROM), a storage drive, an optical disc, and the like. As described in detail below, machine-readable storage medium 508 may be encoded with executable instructions for prioritizing data. For example, the machine-readable storage medium 508 may be encoded with executable instructions for prioritizing individual updates.

[0056] In some examples, a NIC 510 may connect computing system 502 to a network 512. For example, the NIC 510 may connect computing system 502 to a local network 512, a virtual private network (VPN), or the Internet. In some examples, the NIC may include an Ethernet controller. In some examples, the Ethernet controller can be leveraged on a field programmable gate array (FPGA) to provide a connection of the pipeline stages using the transmission control protocol (TCP).

[0057] In some examples, the DMA engine 514 may be an embedded DMA controller. The DMA engine can be used to transport data without accessing the processor 504. For example, the direct memory access (DMA) engine 514 may be used to retrieve data from an FPGA-local dynamic random access memory (DRAM) into block random access memory (BRAM) buffers on a field programmable gate array (FPGA). In some examples, the DMA engine 514 may be used to retrieve an individual update 113 that is to be placed in the priority queue.

[0058] In examples, the priority processor 516 may be an Application Specific Integrated Circuit (ASIC), a Field Programmable Gate Array (FPGA), or other type of specialized processor designed to perform the techniques described herein. For example, an FPGA may be programmed to efficiently prioritize updates 113 in a pipelined array as discussed in Figs. 3A-3F above. For example, the priority processor 516 may receive a first update 113 from the processor and output a second update residing at a root position of the priority queue 104 and send the

second update to the processor and/or the memory and enter the first update at the root position of the priority queue 104. The priority processor 516 may also swap the first update residing at the root position of the priority queue 104 with a third update residing at a second position of the priority queue 104 based on a comparison of a first priority value associated with the first update and a second priority value associated with the third update. In some examples, the priority processor 516 may be further configured to swap the first update residing at the second position of the priority queue 104 with at least a fourth update residing at least a third position of the priority queue 104 based on a comparison of a third priority value associated with the fourth update and the first priority value associated with the first update. In some examples, the swapping between pairs of consecutive odd and even level updates may be executed concurrently.

[0059] In some examples, the priority queue 104 is a data structure that may receive updates 113 for sorting according to a priority. In some examples, the priority queue 104 may be located on priority processor 516. For example, memory 506 may be a memory associated with priority processor 516. In some examples, the priority queue 104 may be located on storage device 508.

[0060] The block diagram of Fig. 5 is not intended to indicate that the computing device 502 is to include all of the components shown in Fig. 5. Further, the computing device 502 may include any number of additional components not shown in Fig. 5, depending on the details of the specific implementation.

[0061] Fig. 6 is a drawing of an example machine-readable storage medium 600 that may be used to update data. Machine-readable storage medium 600 is connected to processor 602 via bus 604. Machine-readable storage medium 600 also contains data fetch module 606, a priority module 608, and a merge module 610. The machine-readable medium is generally referred to by the reference number 600. The machine-readable medium 600 may comprise Random Access Memory (RAM), a hard disk drive, an array of hard disk drives, an optical drive, an array of optical drives, a non-volatile memory, a Universal Serial Bus (USB) flash drive, a DVD, a CD, and the like. In one implementation of the present techniques, the machine-readable medium 600 may be accessed by a processor 602 over a computer bus 604.

[0062] The various software components discussed herein may be stored on the tangible, non-transitory machine-readable medium 600 as indicated in Fig. 6. For

example, a first block 606 may include a data fetch module 606 to retrieve an update. For example, the update can be an individual update 113 from an SCU 110. A second block 608 may include a priority module to order the updates by priority value via a priority queue. For example, the priority value can be a primary key 114 associated with each individual update 113. The priority module 608 may further also receive a higher priority update from the priority queue. A third block 610 may include a merge module 610 to merge the higher priority update with a corresponding database row. In some examples, merging the higher priority update may include instructions to update multiple fields of the corresponding database row based on a timestamp comparison. In some examples, the merge module 610 may receive additional higher priority updates from the priority queue. In some examples, the merge module 610 may merge the higher priority updates with the corresponding database row. In some examples, the merge module 610 may also send the merged database row to a memory after receiving a higher priority update corresponding to a different database row. In some examples, the instructions to update the multiple fields are to be executed in parallel. In some examples, the merge module 610 may store the merged database row to a storage device.

[0063] Although shown as contiguous blocks, the software components may be stored in any order or configuration. For example, if the computer-readable medium 600 is a hard drive, the software components may be stored in non-contiguous, or even overlapping, sectors.

[0064] The present techniques are not restricted to the particular details listed herein. Indeed, those skilled in the art having the benefit of this disclosure will appreciate that many other variations from the foregoing description and drawings may be made within the scope of the present techniques.

CLAIMS

What is claimed is:

1. A computing system for updating data, comprising:

a priority processor to:

receive an individual update;

output a prioritized update residing at a root position of a priority queue and replace the prioritized update with a new individual update; and

sort the individual updates in the priority queue based on their priority values; and

a merge processor to:

receive the sorted updates from the priority processor and merge the sorted updates into merged updates.

2. The computing system of claim 1, the individual updates corresponding to database entries to be updated.

3. The computing system of claim 2, further comprising a direct memory access (DMA) engine to retrieve an individual update from a self-consistent update (SCU) to be placed in the priority queue.

4. The computing system of claim 3, the individual update comprising a primary key to be used as the priority value and self-consistent update (SCU) source ID.

5. The computing system of claim 1, further comprising a block random access memory (BRAM) to provide temporary storage for a plurality of individual updates of each SCU.

6. A method for updating data, comprising:

receiving updates;

ordering the updates by priority value via a priority queue;

receiving a higher priority update from the priority queue; and

merging the higher priority update with a corresponding database row.

7. The method of claim 6, merging the higher priority update comprising updating multiple fields of the corresponding database row based on a timestamp comparison.

8. The method of claim 7, wherein the multiple fields are to be updated in parallel.

9. The method of claim 6, further comprising:

receiving additional higher priority updates from the priority queue;

merging the additional higher priority updates with the corresponding database row; and

sending the merged database row to a memory after receiving a higher priority update corresponding to a different database row.

10. The method of claim 9, further comprising storing the merged database row to a memory after receiving a higher priority update corresponding to a different database row.

11. A non-transitory machine-readable storage medium for updating data encoded with instructions executable by a processor, the machine-readable storage medium comprising:

instructions to retrieve updates;

instructions to order the updates by priority value via a priority queue;

instructions to receive a higher priority update from the priority queue; and

instructions to merge the higher priority update with a corresponding database row.

12. The non-transitory machine-readable storage medium of claim 11, merging the higher priority update further comprising instructions to update multiple fields of the corresponding database row based on a timestamp comparison.

13. The non-transitory machine-readable storage medium of claim 12, wherein the instructions to update the multiple fields are to be executed in parallel.

14. The non-transitory machine-readable storage medium of claim 11, further comprising:

instructions to receive additional higher priority updates from the priority queue to generate a merged database row;

instructions to merge the higher priority updates with the corresponding database row; and

instructions to send the merged database row to a memory after receiving a higher priority update corresponding to a different database row.

15. The non-transitory machine-readable storage medium of claim 11, further comprising instructions to send the merged database row to a memory after receiving a higher priority update corresponding to a different database row.

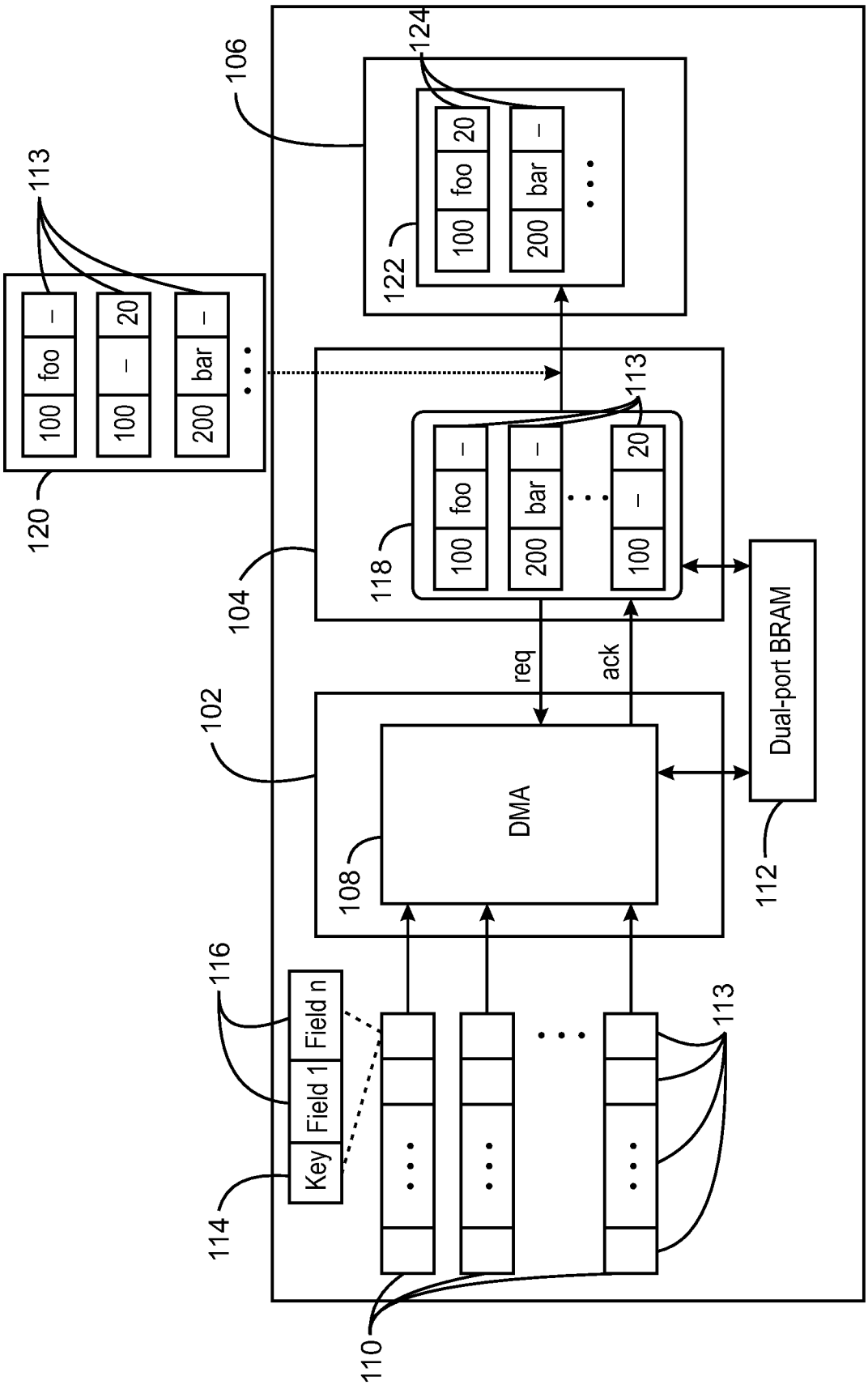
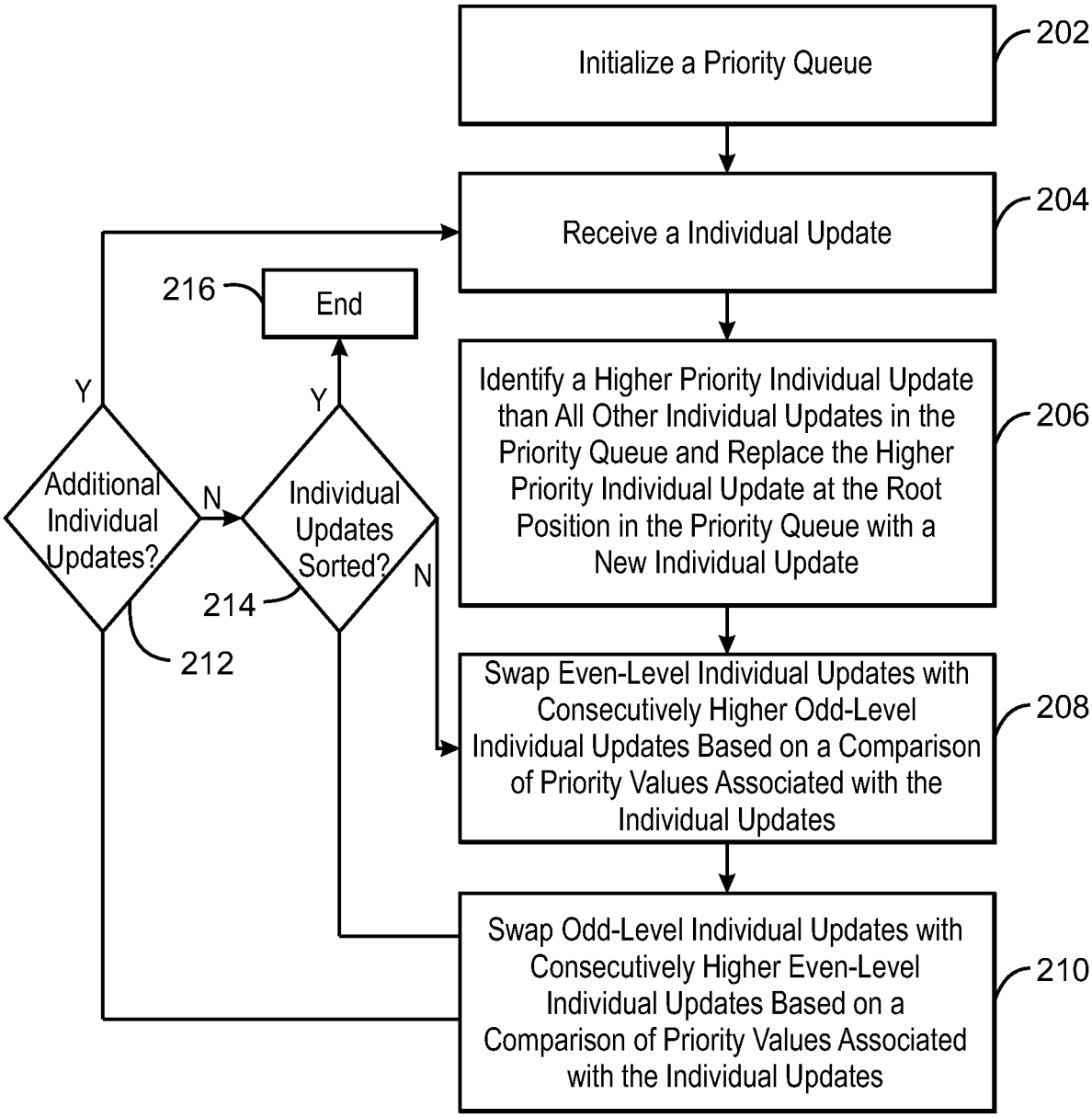
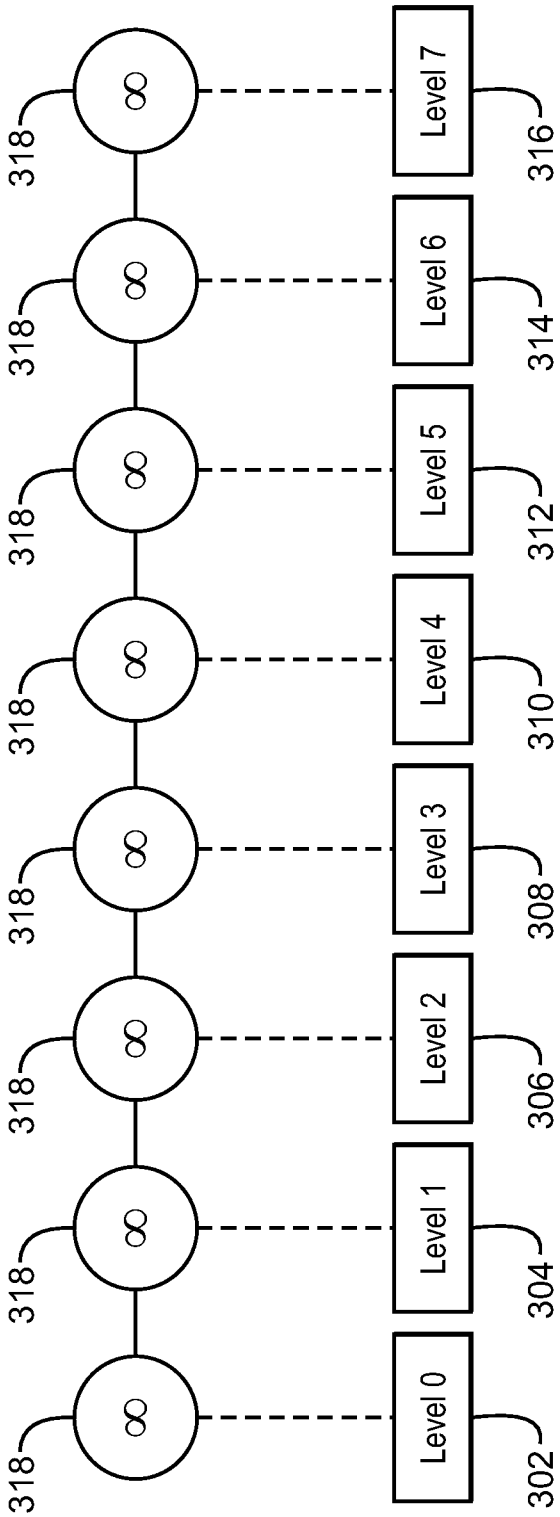

100
106
124
122
100
foo
20
200
bar
113
120
104
118
req
ack
102
DMA
108
Dual-port BRAM
112
116
Key
Field 1
Field n
114
110

FIG. 1


Initialize a Priority Queue
202
Receive a Individual Update
204
Identify a Higher Priority Individual Update than All Other Individual Updates in the Priority Queue and Replace the Higher Priority Individual Update at the Root Position in the Priority Queue with a New Individual Update
206
Swap Even-Level Individual Updates with Consecutively Higher Odd-Level Individual Updates Based on a Comparison of Priority Values Associated with the Individual Updates
208
Swap Odd-Level Individual Updates with Consecutively Higher Even-Level Individual Updates Based on a Comparison of Priority Values Associated with the Individual Updates
210
Individual Updates Sorted?
214
Y
N
Additional Individual Updates?
212
Y
N
End
216

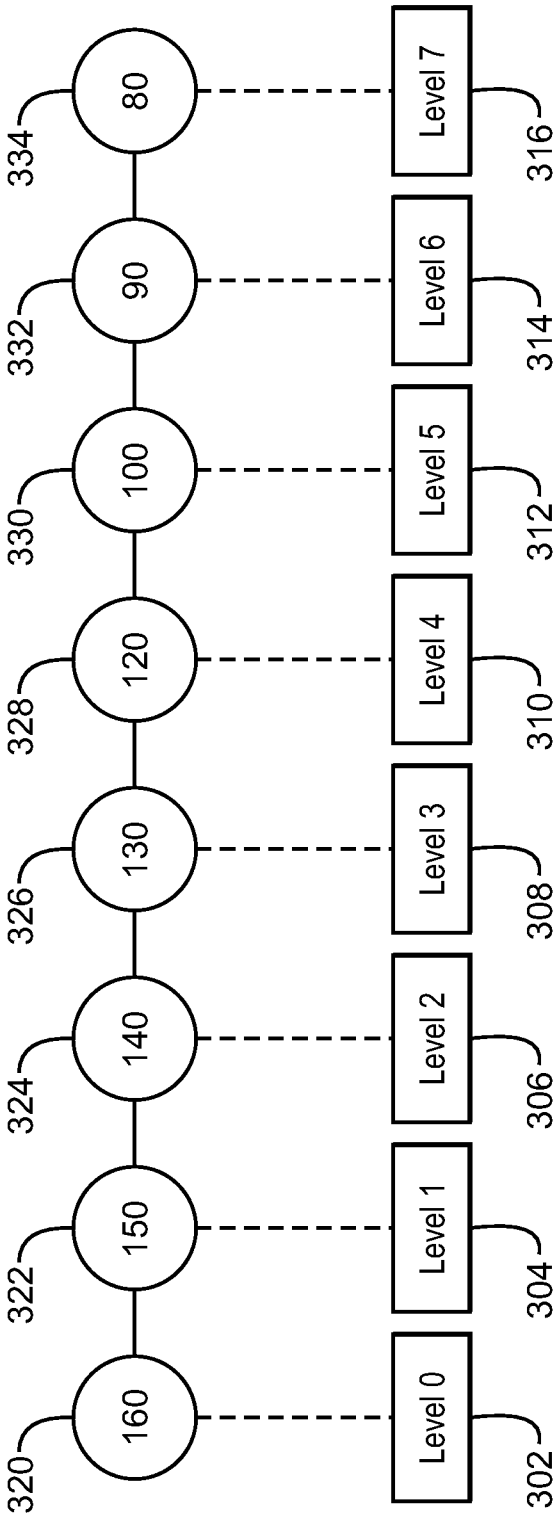
200

FIG. 2


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
302
304
306
308
310
312
314
316
318
∞

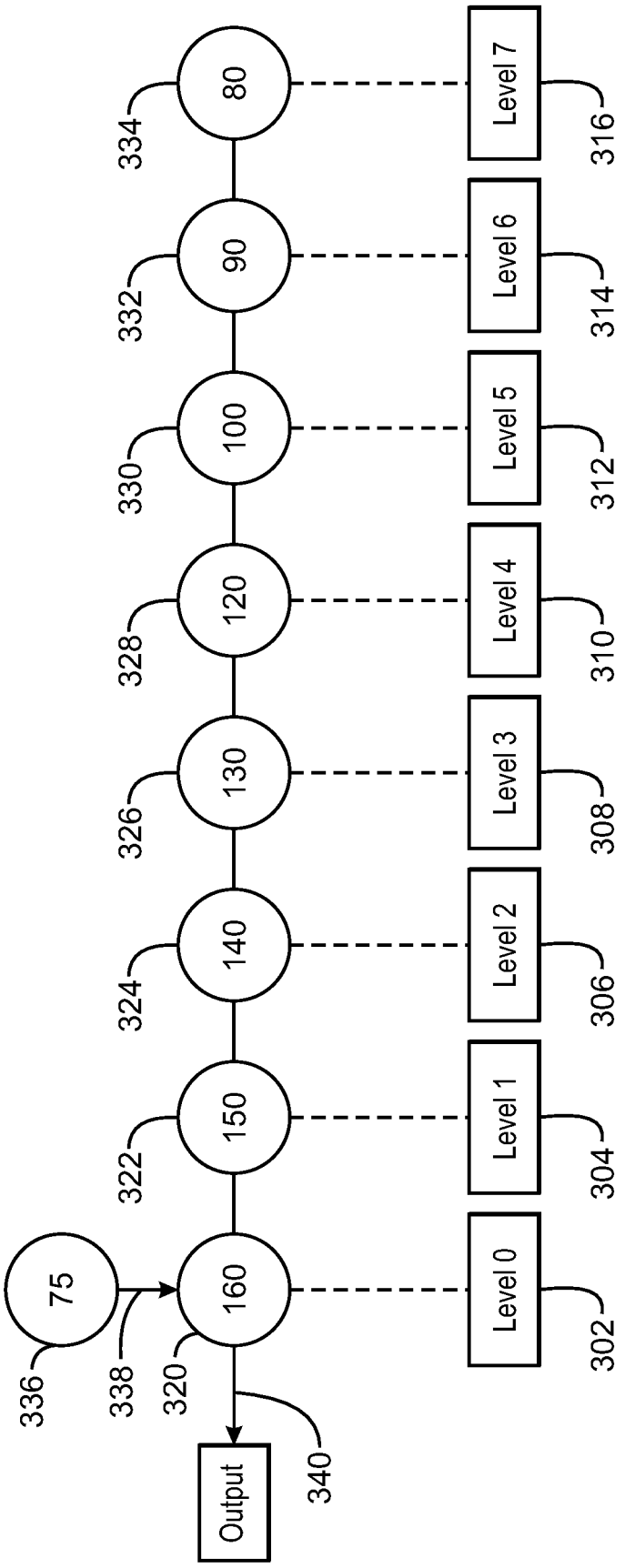
300A

FIG. 3A


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
160
150
140
130
120
100
90
80
302
304
306
308
310
312
314
316
320
322
324
326
328
330
332
334

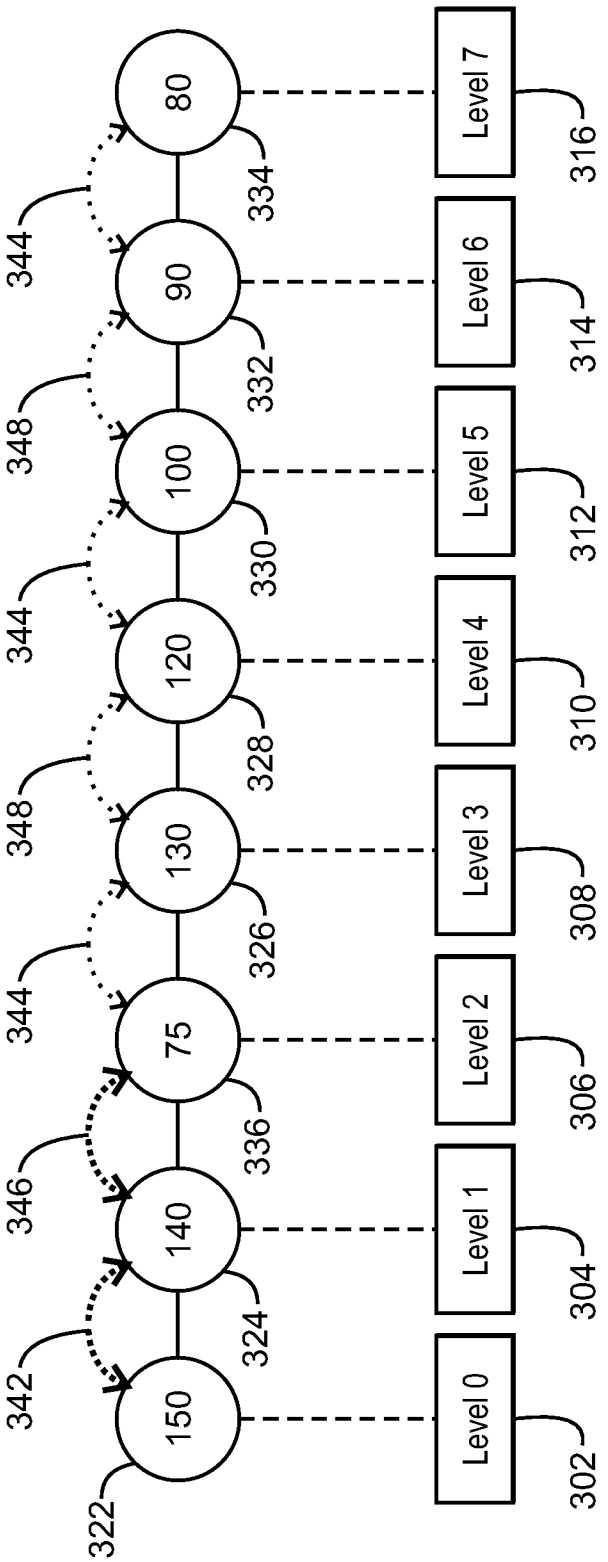
300B

FIG. 3B


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
302
304
306
308
310
312
314
316
160
150
140
130
120
100
90
80
322
324
326
328
330
332
334
75
336
338
320
340
Output

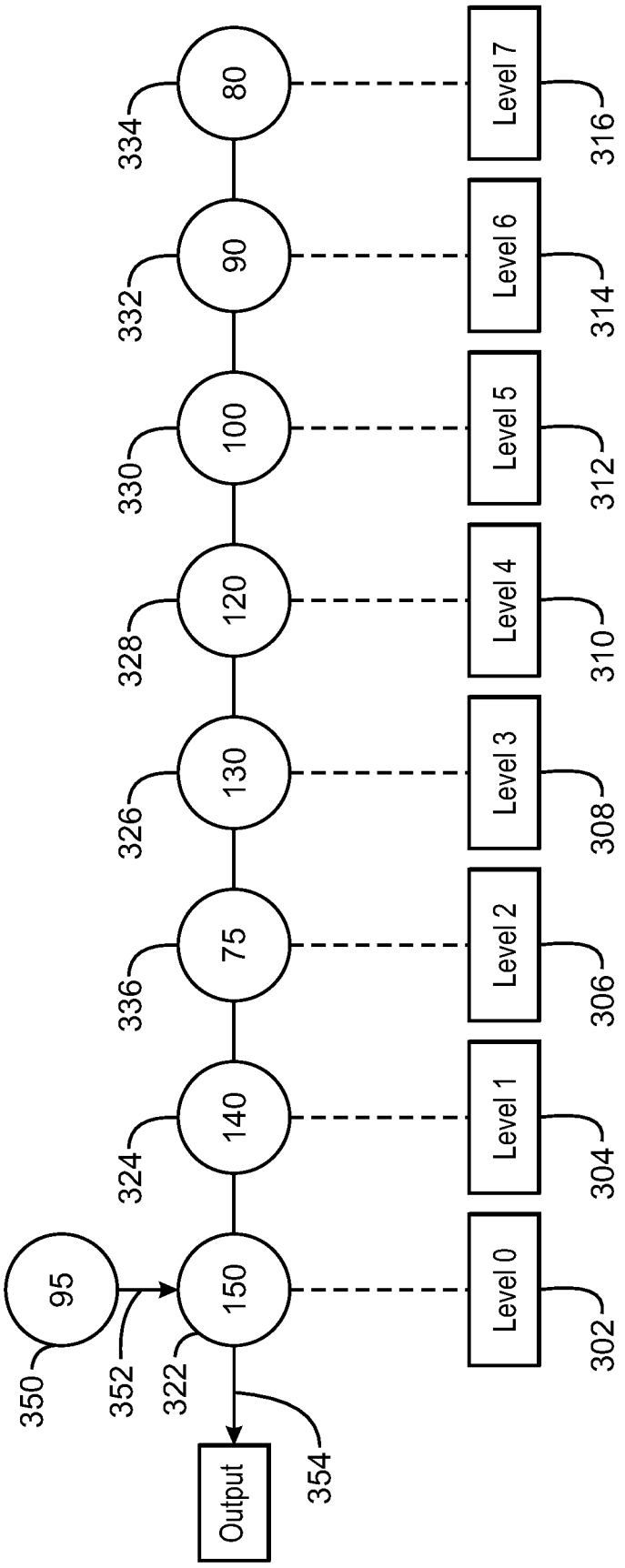
300C

FIG. 3C


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
150
140
75
130
120
100
90
80
322
324
326
328
330
332
334
336
342
344
346
348
302
304
306
308
310
312
314
316

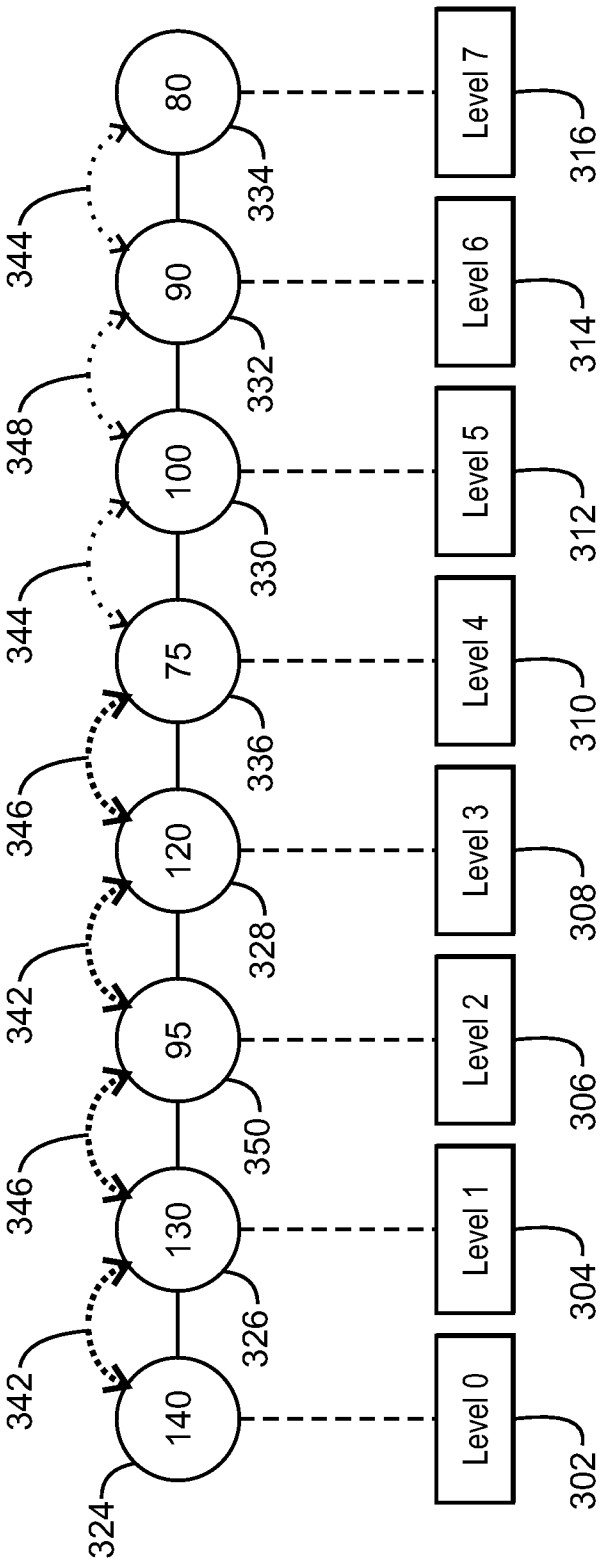
300D

FIG. 3D


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
302
304
306
308
310
312
314
316
150
140
75
130
120
100
90
80
324
336
326
328
330
332
334
95
350
352
322
354
Output

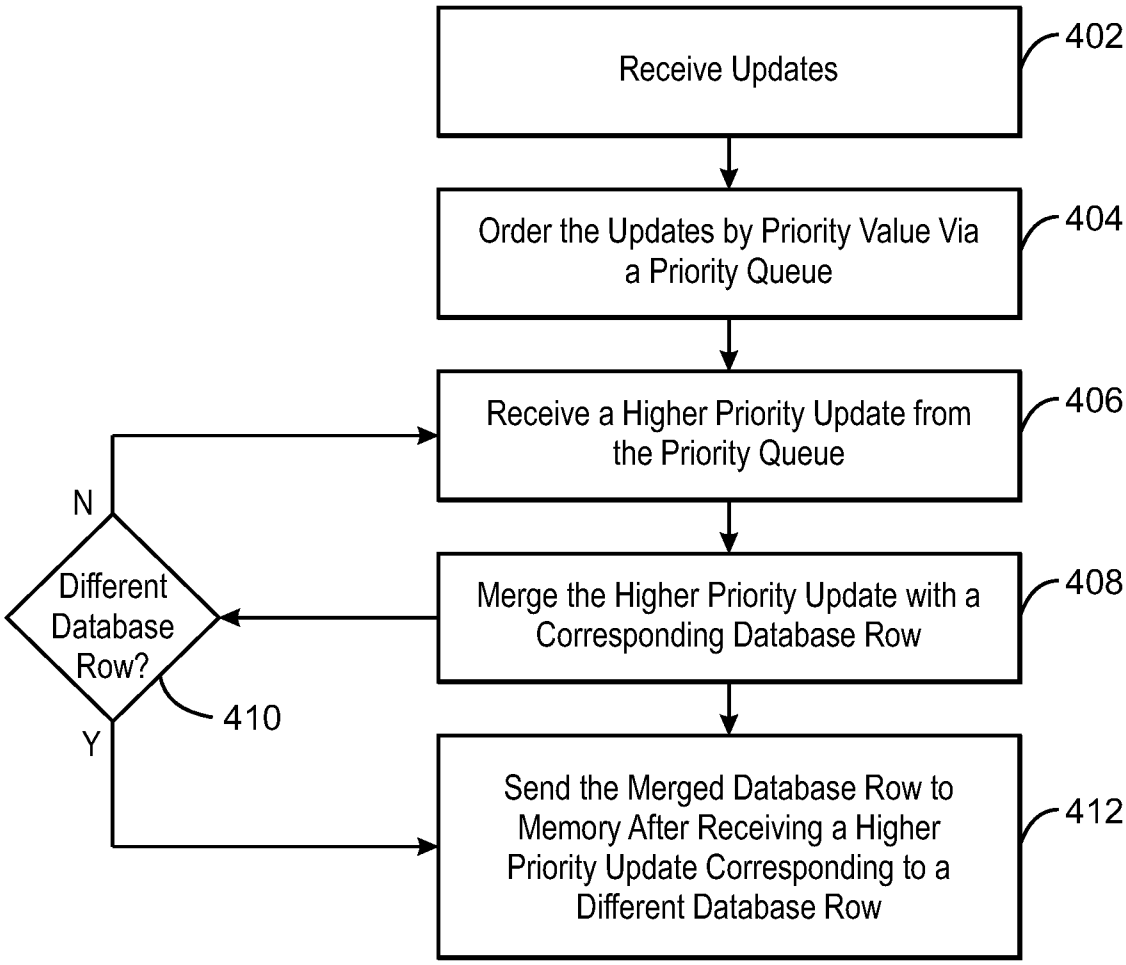
300E

FIG. 3E


Level 0
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Level 7
302
304
306
308
310
312
314
316
140
130
95
120
75
100
90
80
324
326
328
330
332
334
336
342
344
346
348
350

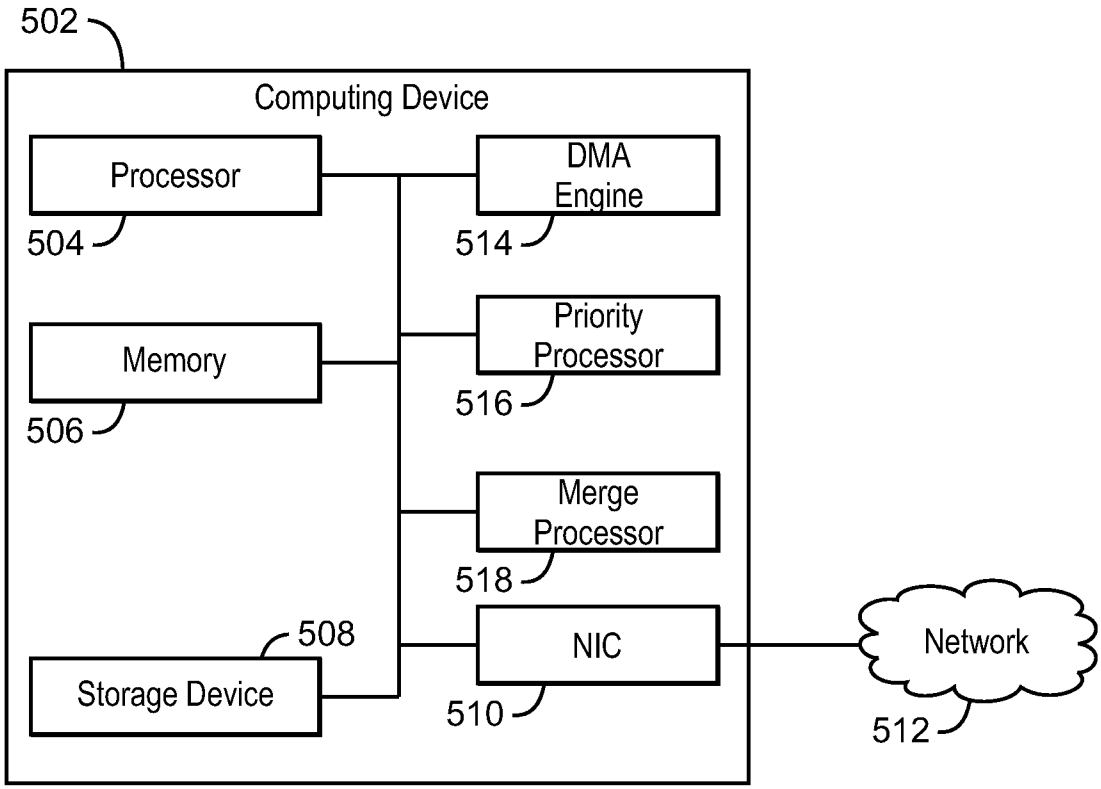
300F

FIG. 3F


Receive Updates
402
Order the Updates by Priority Value Via a Priority Queue
404
Receive a Higher Priority Update from the Priority Queue
406
Merge the Higher Priority Update with a Corresponding Database Row
408
Different Database Row?
410
N
Y
Send the Merged Database Row to Memory After Receiving a Higher Priority Update Corresponding to a Different Database Row
412

400

FIG. 4


502
Computing Device
Processor
504
DMA Engine
514
Memory
506
Priority Processor
516
Merge Processor
518
508
Storage Device
NIC
510
Network
512

500

FIG. 5

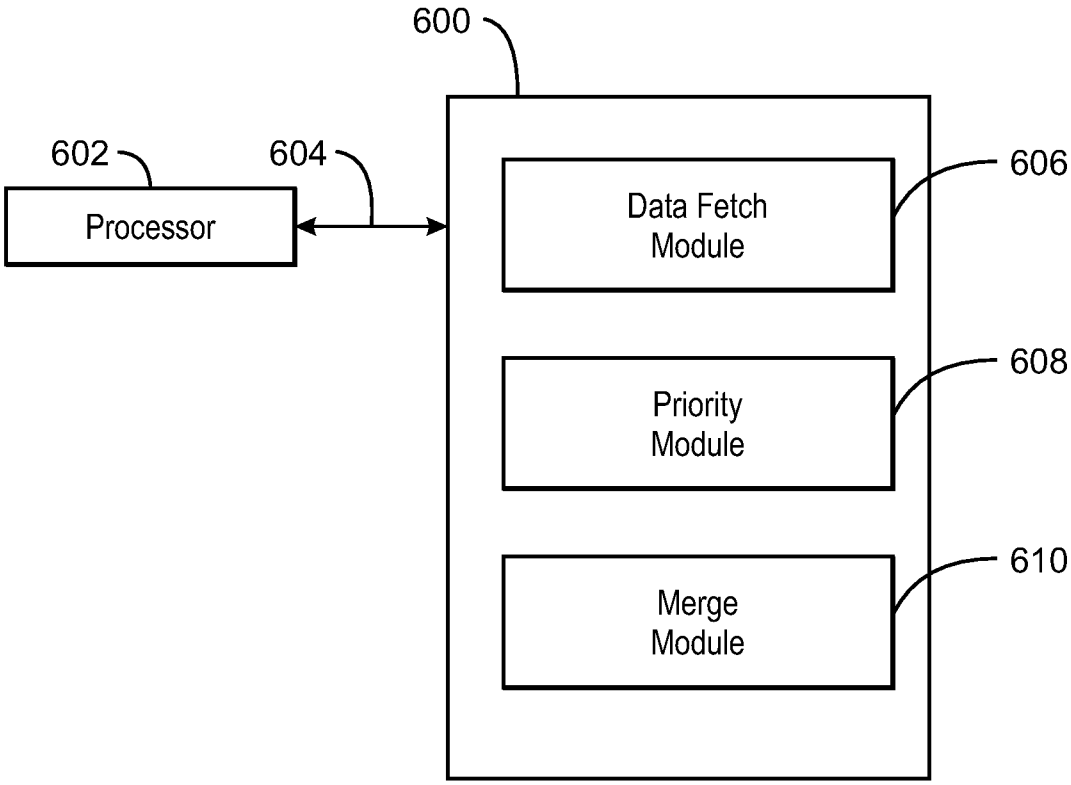

600
602
604
Processor
Data Fetch Module
606
Priority Module
608
Merge Module
610

FIG. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/049264

A. CLASSIFICATION OF SUBJECT MATTER

G06F 17/00(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 17/00; G06F 17/30; G06F 7/00

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: database, update, entry, priority queue, key value, sort, merge, and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2009-0259617 A1 (COWNIE, RICHARD CHARLES et al.) 15 October 2009 See paragraphs [0023], [0026], [0030]-[0034], [0061]-[0062], and [0065]; claims 1 and 6; and figures 1-2 and 6.	1-6,9-11,14-15
A		7-8,12-13
A	US 2012-0254173 A1 (GRAEFE, GOETZ) 04 October 2012 See paragraphs [0015], [0032]-[0034], [0037]-[0038], and [0050]-[0053]; and figures 2a-2b and 5.	1-15
A	US 2010-0281013 A1 (GRAEFE, GOETZ) 04 November 2010 See paragraphs [0041] and [0047]-[0050]; and figure 6.	1-15
A	US 7,016,914 B2 (NAYAK, TAPAS K.) 21 March 2006 See column 13, line 24 - column 15, line 11; and figures 9-10c.	1-15
A	US 2013-0226967 A1 (GROSS, JOHN N et al.) 29 August 2013 See paragraphs [0363]-[0365] and figure 17A.	1-15

☐ 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
30 March 2015 (30.03.2015)	**31 March 2015 (31.03.2015)**

Name and mailing address of the ISA/KR
International Application Division
Korean Intellectual Property Office
189 Cheongsa-ro, Seo-gu, Daejeon Metropolitan City, 302-701, Republic of Korea
Facsimile No. ++82 42 472 7140

Authorized officer
NHO, Ji Myong
Telephone No. +82-42-481-8528

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2014/049264

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2009-0259617 A1	15/10/2009	None	
US 2012-0254173 A1	04/10/2012	None	
US 2010-0281013 A1	04/11/2010	None	
US 7016914 B2	21/03/2006	US 2003-0229626 A1	11/12/2003
		US 2005-0251526 A1	10/11/2005
		US 2006-0010146 A1	12/01/2006
		US 7185019 B2	27/02/2007
		US 7590645 B2	15/09/2009
US 2013-0226967 A1	29/08/2013	None	