(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 2016/018399 A1

(51) International Patent Classification:
G06F 7/06 (2006.01)

(21) International Application Number:
PCT/US2014/049262

(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, Charles, B., III**; 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))*

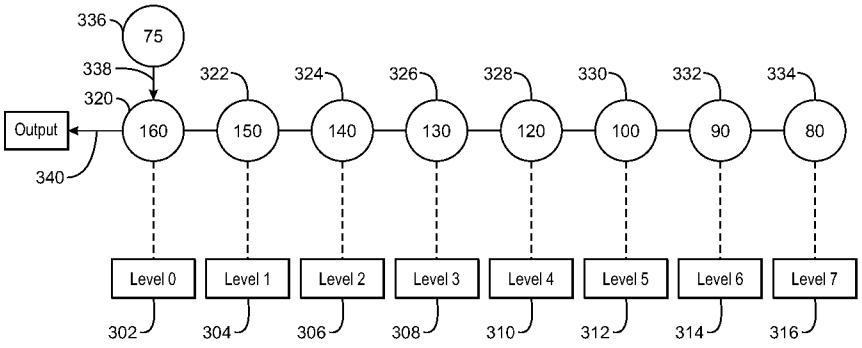
— *as to applicant's entitlement to apply for and be granted a patent (Rule 4.17(ii))*

Published:

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

WO 2016/018399 A1

(54) Title: PRIORITIZATION PROCESSING



300C

FIG. 3C

(57) Abstract: Techniques are described in which a data entry is received at a priority processor. A higher priority entry than all other entries in a priority queue is identified and the higher priority entry is replaced at a root position in the priority queue with the data entry. An even-level entry is also swapped with a consecutively higher odd-level entry based on a comparison of priority values associated with the entries.

PRIORITIZATION PROCESSING

BACKGROUND

[0001] In computing, there are several examples of processes that benefit from prioritizing data. Data may be prioritized using values that indicate the relative priority of individual data.

BRIEF DESCRIPTION OF THE DRAWINGS

[0002] Various features of the invention will become apparent from the following description of examples of the invention, 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 computing device to prioritize data;

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

[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 entries with priority values displayed;

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

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

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

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

[0011] Fig. 4 is a drawing of an example machine-readable storage medium that can be used to prioritize data.

DETAILED DESCRIPTION

[0012] There are processes in computing environments that may benefit from prioritizing data. Data may be prioritized using values that indicate the relative

priority of individual data. For example, packets traveling through routers may be assigned Quality-of-Service (QoS) priority values depending on the type of data they are carrying. In this way, applications with high latency sensitivity such as voice-over-Internet-protocol (VoIP) may experience less latency due to wait times at the router. 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.

[0013] Improved prioritization of data flows may be achieved using priority queues. As used herein, priority queues refer to data structures that are used to prioritize 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 a data packet with a higher Quality of Service (QoS) priority number compared to the other elements in a queue, or a database entry with a higher primary index number than other database entries 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.

[0014] Traditional software implementations use a heap to implement a priority queue. For example, a heap stores its internal nodes in a binary tree structure and maintains the property that any parent node is larger than both of its children nodes. While a heap takes constant time to check for a priority element, it takes a heap a logarithmic time to insert, replace, or remove items. Each time an insertion, replacement, or removal occurs within a heap, the heap fixes the binary tree to regain the properties of a heap. Therefore, tasks that repeatedly insert and/or remove elements from the priority queue are bound by logarithmic time.

[0015] Systolic arrays may also be used to prioritize data. A systolic array is a data structure that can pair two arrays together. For example, two arrays A and B, can be of equal size N, resulting in a total space requirement of 2N. Items may be kept in sorted order in a first array A, and when items are inserted, they may be replaced in the B array. The items may then be compared across B and A arrays to identify if items are to be swapped in the sorted array A to regain ordering, or transferred to the B array to be shifted down. Across clock cycles, array B may be

shifted down by one element per clock cycle. However, systolic arrays again require a total space of 2N and are therefore not space efficient.

[0016] Described herein are techniques relating to the prioritization of data using a specialized priority processor in a computing device. 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. In some examples, a priority processor of the computing device may use replace and delete functions to prioritize data in a pipelined array associated with a priority queue as described in Figs. 3A-3F below. An example of such computing device is shown in Fig. 1.

[0017] Fig. 1 is a block diagram of an example computing device 102 to prioritize data. The computing device 102 may include a processor 104, memory 106, a machine-readable storage 108, a network interface card (NIC) 110 to connect computing system 102 to network 112, a priority processor 114, and a priority queue 116.

[0018] In some examples, the processor 104 may be a main processor that is adapted to execute the stored instructions. The processor 104 may be a single core processor, a multi-core processor, a computing cluster, or any number of other configurations. The processor 104 may be implemented as Complex Instruction Set Computer (CISC) or Reduced Instruction Set Computer (RISC) processors, x86 Instruction set compatible processors, multi-core, or any other microprocessor or central processing unit (CPU).

[0019] In some examples, the memory device 106 may include random access memory (e.g., SRAM, 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 114.

[0020] In some examples, machine-readable storage 108 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 108 may be encoded with executable instructions for prioritizing data.

[0021] In some examples, a NIC 110 may connect computing system 102 to a network 112. For example, the NIC 110 may connect computing system 102 to a local network 112, a virtual private network (VPN), or the Internet. In some examples, the computing device may be a router 102 and the NIC 110 may also represent ports 110 by which router 102 is connected to one or more computing devices in network 112.

[0022] In examples, the priority processor 114 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 process data in a pipelined array as discussed in Figs. 3A-3F below. For example, the priority processor 114 may receive a first data entry from the processor and output a second data entry residing at a root position of the priority queue and send the second data entry to the processor and/or the memory and enter the first data entry at the root position of the priority queue. The priority processor 114 may also swap the first data entry residing at the root position of the priority queue with a third data entry residing at a second position of the priority queue based on a comparison of a first priority value associated with the first data entry and a second priority value associated with the third data entry. In some examples, the priority processor 114 may be further configured to swap the first data entry residing at the second position of the priority queue with at least a fourth data entry residing at least a third position of the priority queue based on a comparison of a third priority value associated with the fourth data entry and the first priority value associated with the first data entry. In some examples, the swapping between pairs of consecutive odd and even level entries may be executed concurrently.

[0023] In some examples, the priority queue 116 is a data structure that may receive network packets or database entries for priority sorting. In some examples, the priority queue 116 may be located on memory 106. For example, memory 106 may be a memory associated with priority processor 114. In some examples, the priority queue may be located on storage device 108.

[0024] The block diagram of Fig. 1 is not intended to indicate that the computing device 102 is to include all of the components shown in Fig. 1. Further, the

computing device 102 may include any number of additional components not shown in Fig. 1, depending on the details of the specific implementation.

[0025] Fig. 2 is a process flow diagram illustrating an example method for prioritizing data. The method of Fig. 2 is generally referred to by the reference number 200.

[0026] At block 202, processor 104 initializes a priority processor 114. 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 processor 114. In some examples, the processor 104 may populate a priority processor 114 with values indicating infinity. In some examples, the processor 104 may use lower values to indicate priority and may populate the priority processor 114 with values indicating negative infinity. In both cases, the infinity values serve as initial placeholders in the priority processor 114 that may be replaced by data entries using the replace operation.

[0027] At block 204, the priority queue 116 receives a data entry. In examples, the data entry may be one of many data to be prioritized. For example, the data could be network packets to be routed, or database entries to be updated. The data entry may include the data to be processed or a pointer to the data. For example with regard to network packets, each data entry may include the actual packet of data or a pointer that identifies the packet of data. Each data entry also includes a priority value associated with the data entry. The priority value may be assigned by the processor 104. For example, with respect to network packets, the priority value may be a Quality of Service (QoS) value that indicates a packet belongs to a service with a given priority. In some examples, the priority value may indicate a relative position of a database entry to be updated. For example, related database entries to be updated may be grouped together using the priority value and updated by processor 104 in a more efficient order and/or as a batch.

[0028] At block 206, the priority processor 114 identifies a higher priority entry than all other entries in the priority queue and replaces the higher priority entry at the root position in a priority queue with the data entry. A root position in a priority queue, as used herein, refers to a level in a priority queue that receives new entries and contains higher priority entries. After identifying the higher priority entry, the priority processor 114 may send the identified entry to the processor 104 to indicate the corresponding data packet or database entry. In some examples, the priority

processor 114 may send the identified higher priority entries to a memory for later batch processing by processor 104. In some examples, the root level of the priority queue may contain the higher priority entry after a complete clock cycle. In some examples, the root level may contain the new entry after a complete clock cycle. Thus, by replacing the entry at the root position of the priority queue with a new data entry, the priority processor 114 may identify an entry with a higher priority the rest of the entries in the array.

[0029] At block 208, the priority processor 114 swaps even-level entries with consecutively higher odd-level entries based on a comparison of priority values associated with the entries. For example, given six levels 0-5, levels 0 and 1 may be swapped, levels 2 and 3 may be swapped, and levels 4 and 5 may be swapped. In some examples, two levels are swapped based on their priority values. For example, the levels may be swapped when the higher level entry has a higher priority value. For example, in a priority queue, the higher priority values will be sorted to lower levels. Thus, in the priority queue, if level 0 has a priority value of 5 and level 1 has a priority value of 2, then level 0 will not be swapped after being compared with level 1 because they are already sorted correctly. In some examples, the priority processor 114 may simultaneously swap all the pairs of odd/even levels 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.

[0030] At block 210, the priority processor 114 swaps odd-level entries with consecutively higher even-level entries based on a comparison of priority values associated with the entries. For example, level 1 might be swapped with level 2, level 3 might be swapped with level 4, and so on. In some examples, the levels are swapped according to their priority values. For example, level 1 may be swapped with level 2 if level 1 has a lower priority value than level 2. In some examples, block 210 may be executed at higher levels concurrently with the execution of block 206 at the lower levels.

[0031] In some examples, as indicated by diamond 212, if additional entries are received by the priority queue 116, then method 200 may iterate through additional received entries by cycling through blocks 204-210. If no further additional entries are received, then the method proceeds to diamond 214.

[0032] In some examples, as indicated by diamond 214, if no additional entries are received by priority queue 116, then method 200 may proceed to cycle through

blocks 208-210 until no further swaps are performed because all the entries are sorted. In some examples, the priority processor 114 may be populated with lower priority values to sort the remaining sorted entries and identify higher priority entries in the priority queue. If all the entries are sorted, then the method ends at block 216.

[0033] 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.

[0034] Fig. 3A is a diagram of an example priority processor 114 initialized with initial values. The configuration of the example priority processor 114 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.

[0035] 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 entries and store the rest of the entries in a descending order from left to right. In some examples, the priority processor 114 may use replace and delete functions, and not insert functions. By using the replace and delete functions on the priority processor 114 in parallel on all the levels of the priority queue, rather than using insert functions, the priority processor 114 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 114 may efficiently process data entries regardless of the total amount of entries or the size of the entries to be processed. Furthermore, because a single array is used, the priority processor 114 may use storage space efficiently.

[0036] Fig. 3B is a diagram of an example priority processor fully populated with an ordered array of entries with priority values displayed. The configuration of the example priority processor 114 in Fig. 3B is referred to generally by the reference number 300B. Entries 320-334 correspond to levels 302-316 of the priority processor, respectively.

[0037] In the diagram of Fig. 3B, the corresponding priority values of entries 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 entries 320-334 is from higher to lower. The original order of the entries 320-334 does not matter because of the swapping function as discussed at greater length in Fig. 3C. Thus, the priority processor 114 is able to efficiently sort data entries regardless of their original order.

[0038] Fig. 3C is a diagram of an example priority processor receiving a new entry. The configuration of the priority processor 114 in Fig. 3C is generally referred to by the reference number 300C. In addition, new entry 336 is about to replace entry 320 as shown by arrow 338. Entry 320 is also about to be identified as a higher priority entry and sent to output as shown by arrow 340. In some examples, output may be processor 104 or memory 106.

[0039] In the diagram of Fig. 3C, the fully populated priority processor 114 receives a new data entry 336. The new data entry 336 is received at level 0 302, also referred to herein as the root level 302. In some examples, the priority processor 114 uses the replace function to replace entry 320 at root level 302 with new entry 336 and output entry 320. In some examples, the entry 320 may be output 340 to a processor 104, memory 106, or storage device 108. In some examples, the priority processor 114 may then swap consecutive entries using the replace operation as described in Fig. 3D.

[0040] Fig. 3D is a diagram of an example priority processor 114 having processed a new entry. The configuration of the priority processor 114 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.

[0041] In the diagram of Fig. 3D, entry 336 has shifted two places to the right from root level 302 to level 306. In some examples, the replacement of entry 336 with the original entry at root level 302 and the shifting of entry 336 two levels to the right may be performed by the priority processor 114 within one clock cycle. In some examples, the priority processor 114 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 entry 336 at root level 302 swapping with higher priority entry 322 at level 304. Thus, entry 322 is then placed into root level 302 and entry level 336

takes its place at level 304. Although comparisons are made as indicated by arrows 344, the priority processor 114 does not perform any swaps because the priority values of these entries indicated that they are already ordered in a descending order of priority. In a second set of swap and comparisons, as indicated by arrows 346 and 348, entry 336 of level 304 is then swapped with higher priority entry 324 of level 306. Thus, entry 336 moves up to level 306, and entry 324 moves down to level 304, the final resulting order of the entries shown in the example of 300D.

[0042] Fig. 3E is a diagram of an example priority processor 114 receiving another entry. The configuration of the priority processor 114 in Fig. 3E is generally referred to by the reference number 300E. A new entry 350 is to replace entry 322 as shown by arrow 352. Entry 322 is also to be output by the priority processor 114 as shown by arrow 354.

[0043] In the diagram of Fig. 3E, a new entry 350 is to be added to the pipeline processor configuration of 300D. As in 300C, the new entry 350 is to replace the existing entry 322 of root level 302, the existing entry 322 to be output by the priority processor 114 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.

[0044] Fig. 3F is a diagram of an example priority processor 114 having processed the entries. The configuration of the priority processor 114 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.

[0045] In Fig. 3F, both new entry 350 of 300E and entry 336 of 300C have been shifted up two levels to the right. As discussed in 300D, the priority processor 114 executes two consecutive swaps, however 300F shows two pairs of consecutive swaps. In some examples, more than one entry may simultaneously be swapped with a consecutively higher level entry. For example, in 300F entry 350 of root level 302 was swapped with entry 324 of level 304, and entry 336 of level 306 was swapped with entry 326 of level 308. In some examples, after the even-level entries compared and/or swapped with consecutively higher odd-level entries, then the odd-level entries are compared with the corresponding consecutively higher level even-level entries. For example, in the example of 300F, entry 350 at level 304 was compared and swapped with entry 326 of level 306, and entry 336 of level 308 was

compared and swapped with entry 328 of level 310. Thus, 300F shows the final positions of the two sets of swaps. As entries 330 - 334 are still ordered properly with respect to each other, no swaps included them. However, in some examples, with two additional clock cycles, entry 336 may eventually reach level 316 as it is an entry with a lower priority. Likewise, in some examples, with an additional clock cycle, priority processor 114 may swap entry 350 into level 310 and keep it there until a lower priority entry is introduced at later clock cycles. Processing a pipelined array on a priority processor 114 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.

[0046] 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 data entries may be swapped at the same time with a consecutively higher level, depending on the priority values of the data entries.

[0047] Fig. 4 is a drawing of an example machine-readable storage medium 400 that may be used to prioritize data. Machine-readable storage medium 400 is connected to processor 402 via bus 404. Machine-readable storage medium 400 also contains pipelined array module 406. The machine-readable medium is generally referred to by the reference number 400. The machine-readable medium 400 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 embodiment of the present invention, the machine-readable medium 400 may be accessed by a processor 402 over a computer bus 404.

[0048] The various software components discussed herein may be stored on the tangible, non-transitory machine-readable medium 400 as indicated in Fig. 4. For example, a first block 406 may include a pipelined array module 406 to initialize a priority processor 114 with initial values. In some examples, the priority processor 114 may be an FPGA and/or ASIC. The pipelined array module 406 also contains instructions to send a new data entry to a priority queue 116 and swap an existing

entry at a root 302 in the priority queue 116 with the new data entry and identify a higher priority entry than all other entries in the priority queue 116 and replace the higher priority entry at the root position 302 with the new data entry. The pipelined array module 406 further swaps an even-level entry in the priority queue 116 with a consecutively higher odd-level entry in the priority queue 116 based on a comparison of their priority values. The pipelined array module 406 also swaps an odd-level entry in the priority queue 116 with a consecutively higher even-level entry based on a comparison of their priority values. In some examples, the instructions may include replace and delete operations, and not insert operations. For example, the root entry may be swapped, and the even-level entries and the odd-level entries also swapped, using the replace function. In some examples, the instructions to swap the data entry in the root position of the priority queue and to swap the even-level and odd-level entries and the odd-level with the even-level entries in the priority queue are to be performed by the priority processor 114 in one clock cycle. For example, the pipelined array module 406 may send a new data entry to an FPGA, which then replaces root entry 302 with the new data entry and performs two sets of comparisons and/or swaps and identifies a higher priority entry, all in one clock cycle. In some examples, the priority processor 114 is to perform additional swaps in additional clock cycles until the entries in the priority queue are all sorted. For example, the priority processor may continue to swap even-level entries in the priority queue with consecutively higher odd-level entries in the priority queue based on a comparison of their priority values, and odd-level entries in the priority queue with consecutively higher even-level entries based on a comparison of their priority values, until all the entries in the priority queue are sorted according to their priority values. In some examples, the clock cycle may begin with performing two sets of comparisons and/or swaps as discussed in Figs. 3A-3F, and then the pipelined array module 406 sends a new data entry to replace the higher priority entry at the root position of the priority queue and identify the higher priority entry. In some examples, each set of comparisons and/or swaps may be performed by the priority processor 114 in parallel.

[0049] Although shown as contiguous blocks, the software components may be stored in any order or configuration. For example, if the computer-readable medium 400 is a hard drive, the software components may be stored in non-contiguous, or even overlapping, sectors.

[0050] 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 prioritizing data, comprising:

a priority processor to:

receive a first data entry from another processor,

output a second data entry residing at a root position of a priority queue and send the second data entry to the other processor and/or a memory and enter the first data entry at the root position of the priority queue, and

swap the first data entry residing at the root position of the priority queue with a third data entry residing at a second position of the priority queue based on a comparison of a first priority value associated with the first data entry and a second priority value associated with the third data entry.

2. The computing system of claim 1, the priority processor further configured to swap the first data entry residing at the second position of the priority queue with at least a fourth data entry residing at least a third position of the priority queue based on a comparison of a third priority value associated with the fourth data entry and the first priority value associated with the first data entry.

3. The computing system of claim 2, the computing system comprising a router and the data entry corresponding to a network packet to be routed based on quality of service requirements.

4. The computing system of claim 1, the priority processor comprising a field-programmable gate array (FPGA) and/or an application-specific integrated circuit (ASIC).

5. The computing system of claim 1, the data entry comprising a pointer to a data to be prioritized.

6. A method, comprising:

receiving a data entry at a priority processor;

identifying a higher priority entry than all other entries in a priority queue and swapping the higher priority entry at a root position in the priority queue with the data entry; and

swapping an even-level entry in the priority queue with a consecutively higher odd-level entry in the priority queue based on a comparison of priority values associated with the entries.

7. The method of claim 6, further comprising swapping an odd-level entry in the priority queue with a consecutively higher even-level entry in the priority queue based on a comparison of the priority values associated with the entries.

8. The method of claim 6, the priority processor comprising an FPGA and/or ASIC.

9. The method of claim 7, the swapping to be performed by the priority processor in one clock cycle, the priority processor to swap entries for additional clock cycles until the entries in the priority queue are sorted.

10. The method of claim 6, the priority processor to use replace and delete operations, and not insert operations.

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

instructions to initialize a priority processor with initial values;

instructions to send a new data entry to a priority queue and identify a higher priority entry than all other entries in the priority queue and replace the higher priority entry at the root position with the new data entry;

instructions to swap an even-level entry in the priority queue with a consecutively higher odd-level entry in the priority queue based on a comparison of their priority values; and

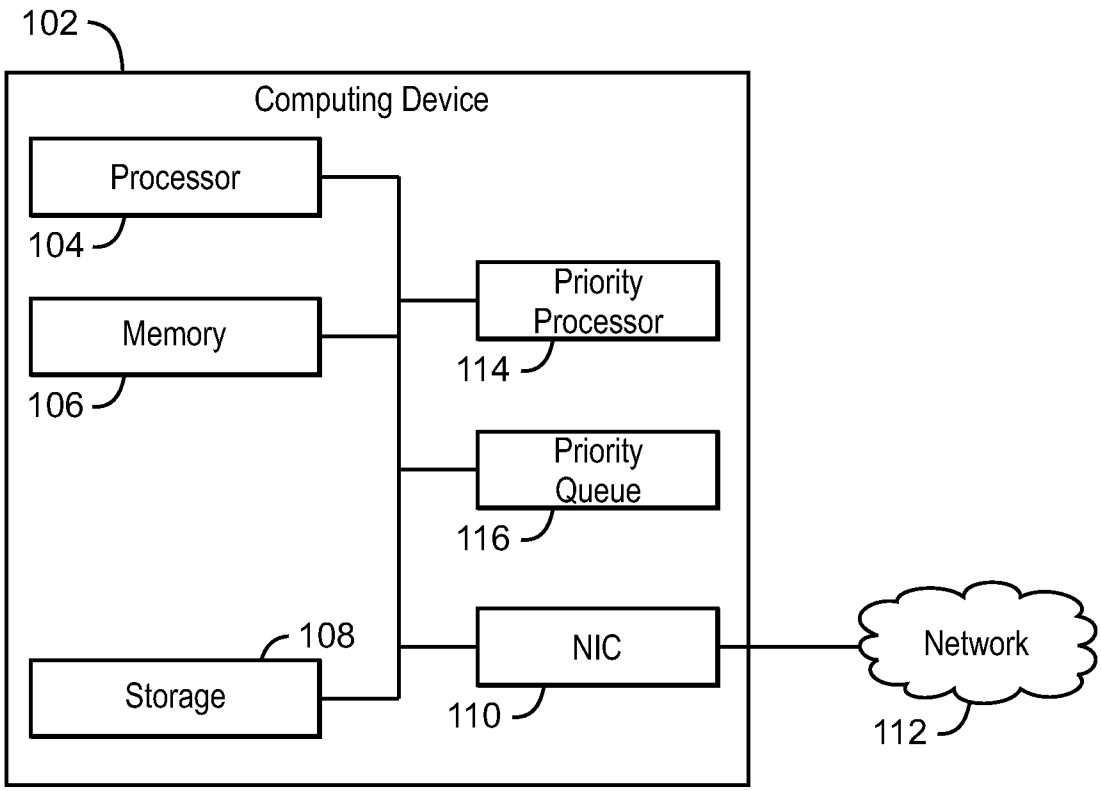
instructions to swap an odd-level entry in the priority queue with a consecutively higher even-level entry based on a comparison of their priority values.

12. The non-transitory machine-readable storage medium in accordance with claim 11, further comprising instructions to continue to swap even-level entries in the priority queue with consecutively higher odd-level entries in the priority queue based on a comparison of their priority values, and swap odd-level entries in the priority queue with consecutively higher even-level entries based on a comparison of their priority values, until all the entries in the priority queue are sorted according to their priority values.

13. The non-transitory machine-readable storage medium in accordance with claim 11, the priority processor comprising an FPGA and/or an ASIC.

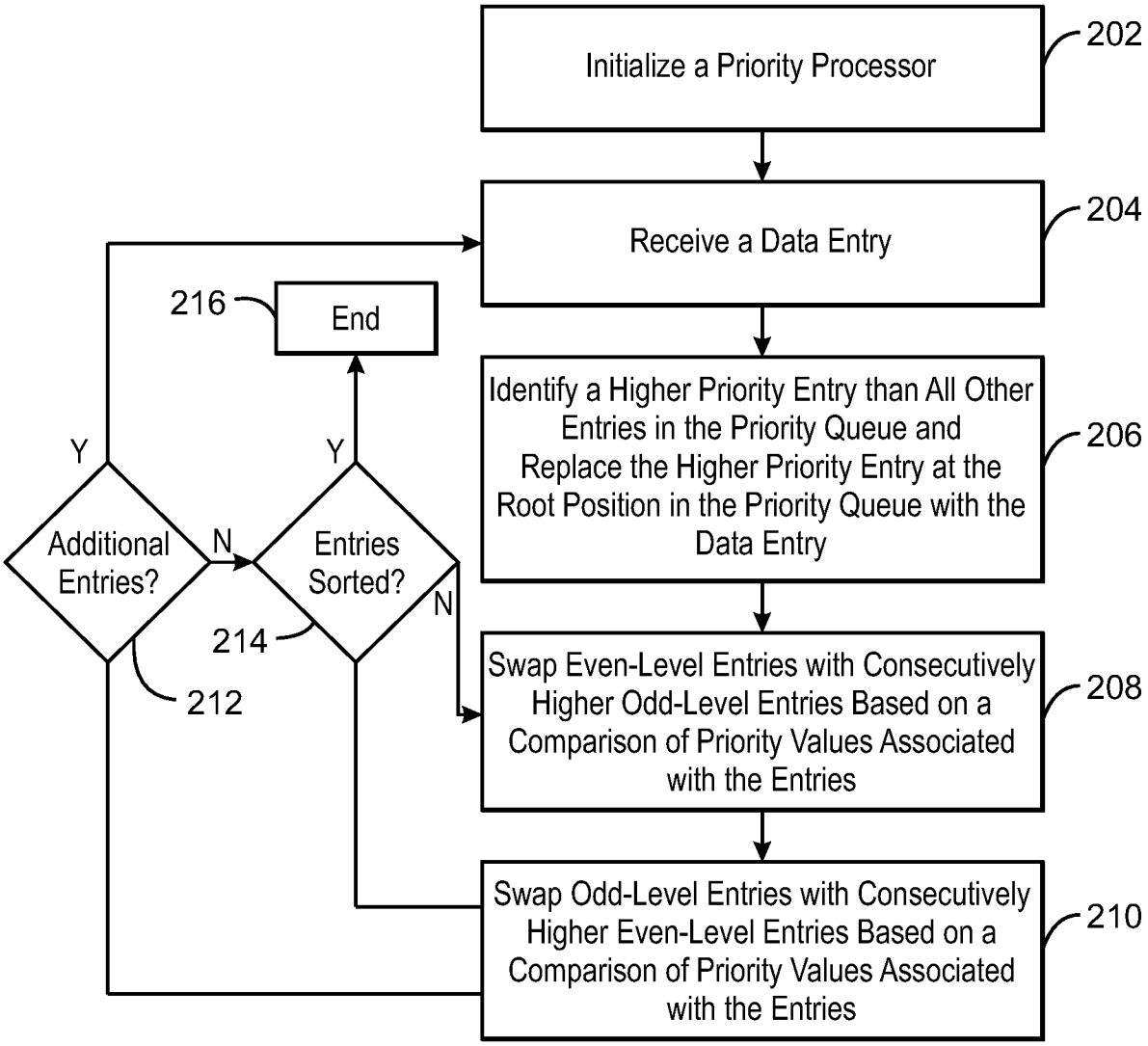
14. The non-transitory machine-readable storage medium in accordance with claim 11, the instructions to send the data and to swap the entries to be performed by the priority processor in one clock cycle, the priority processor to perform additional swaps in additional clock cycles until the entries in the priority queue are sorted.

15. The non-transitory machine-readable storage medium in accordance with claim 11, the instructions to include replace and delete operations, and not insert operations.


102
Computing Device
Processor
104
Memory
106
Priority Processor
114
Priority Queue
116
108
Storage
NIC
110
Network
112

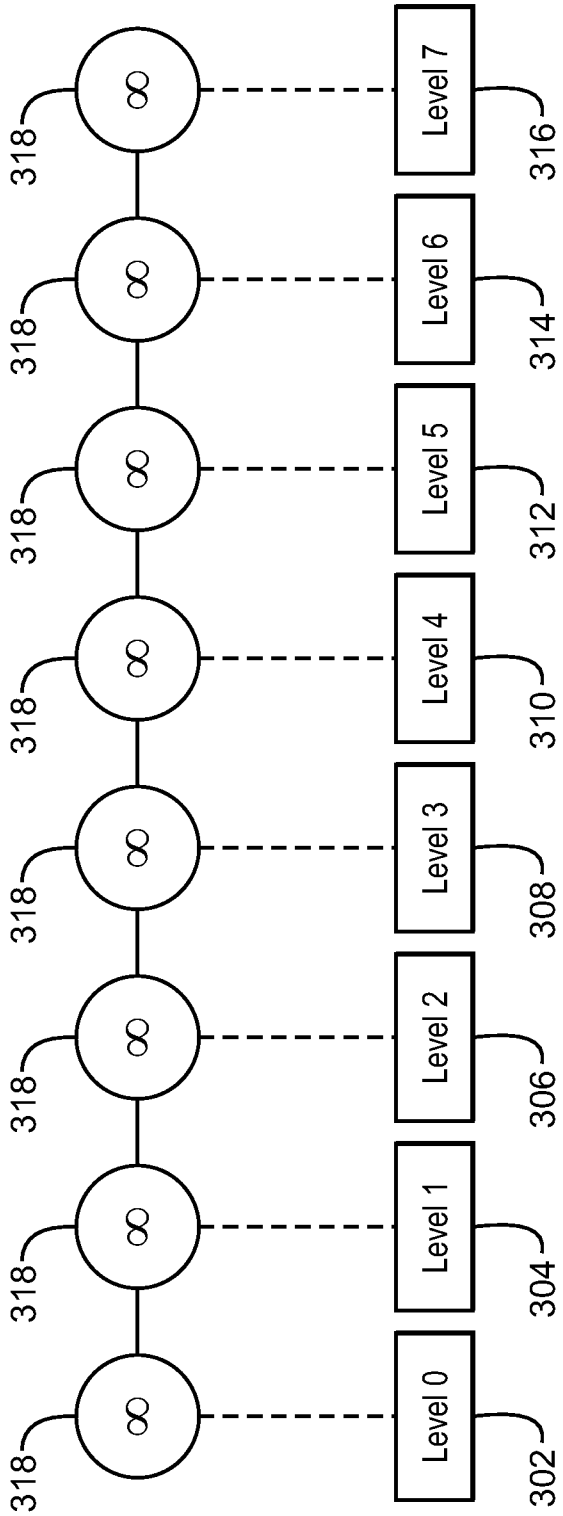
100

FIG. 1


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

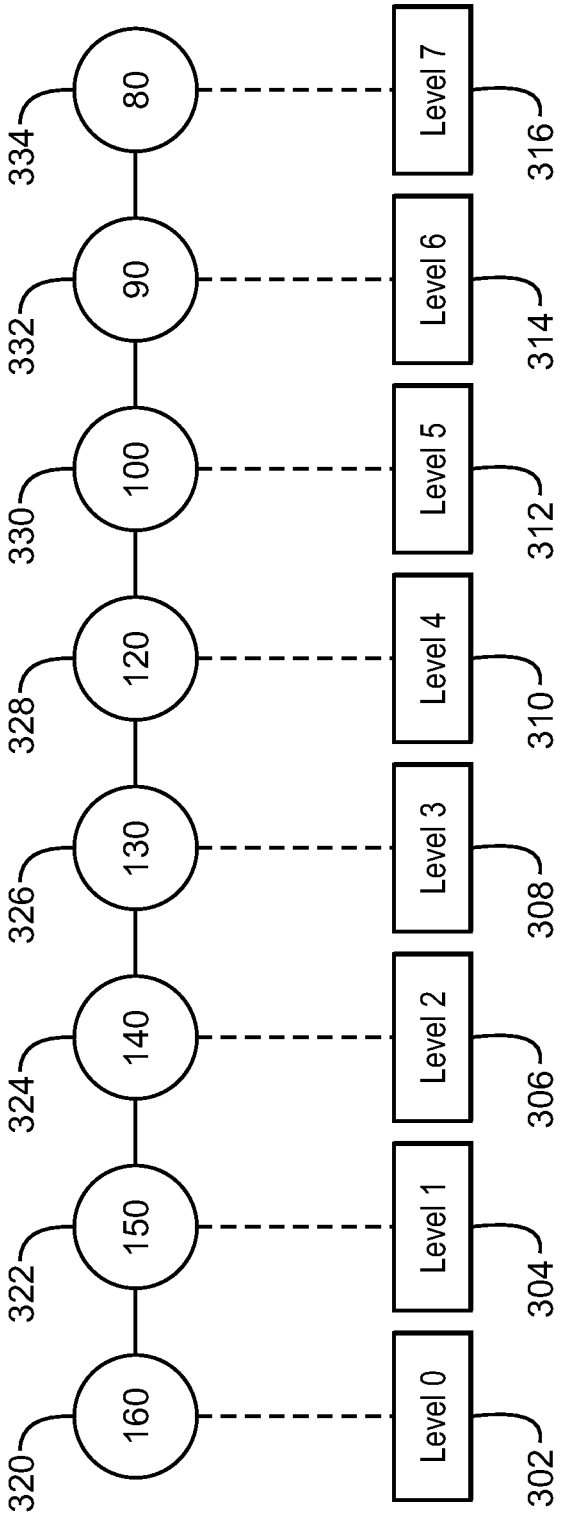
200

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

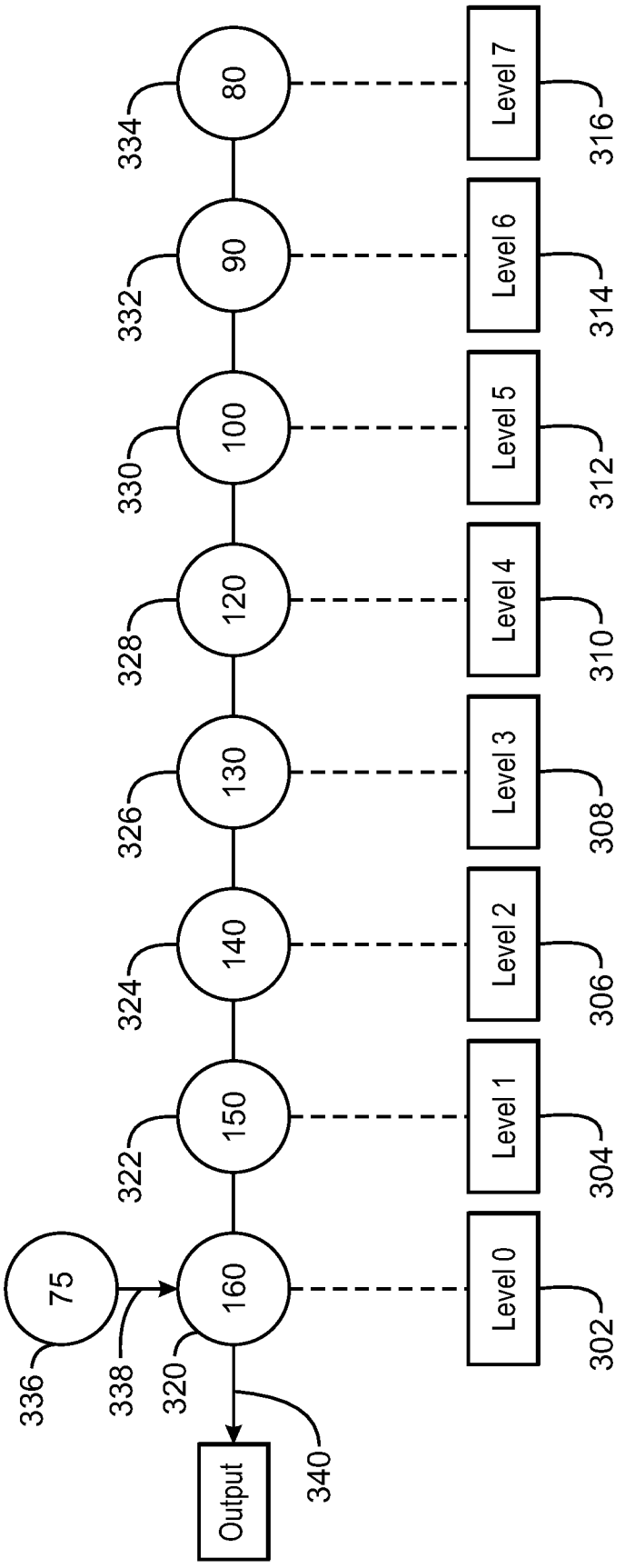
300A

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

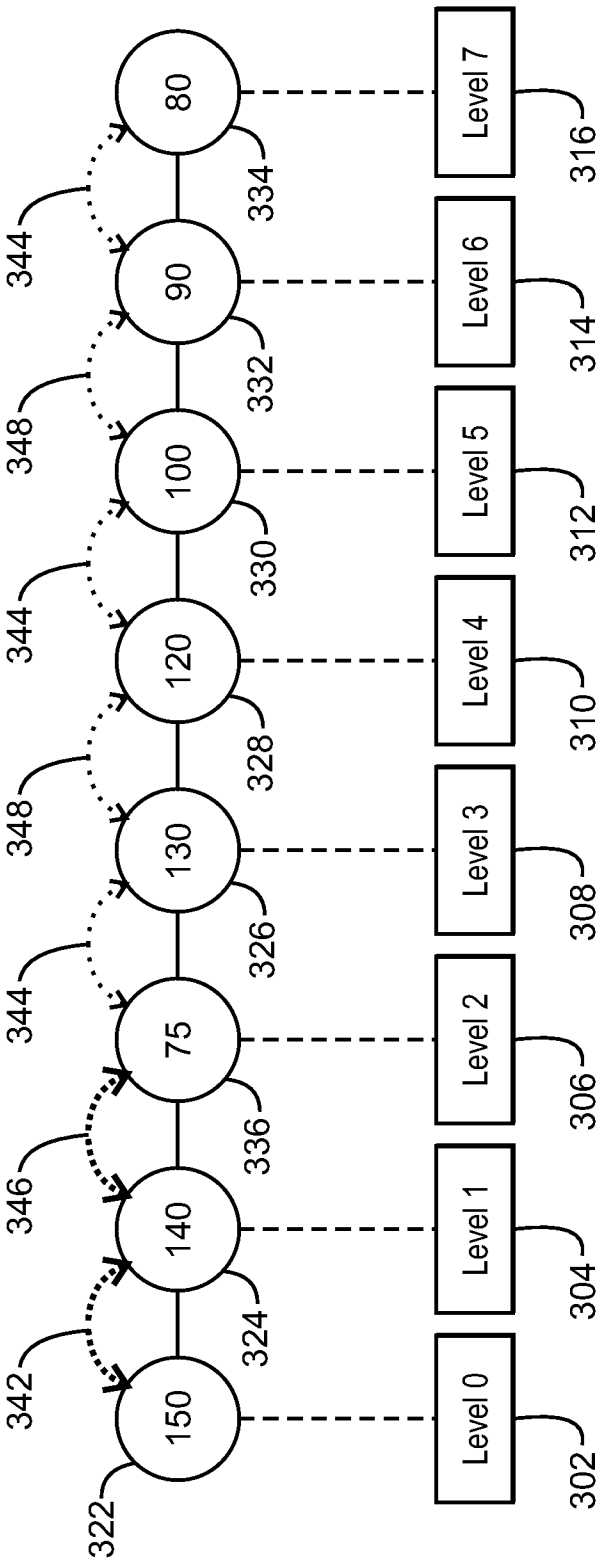
300B

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

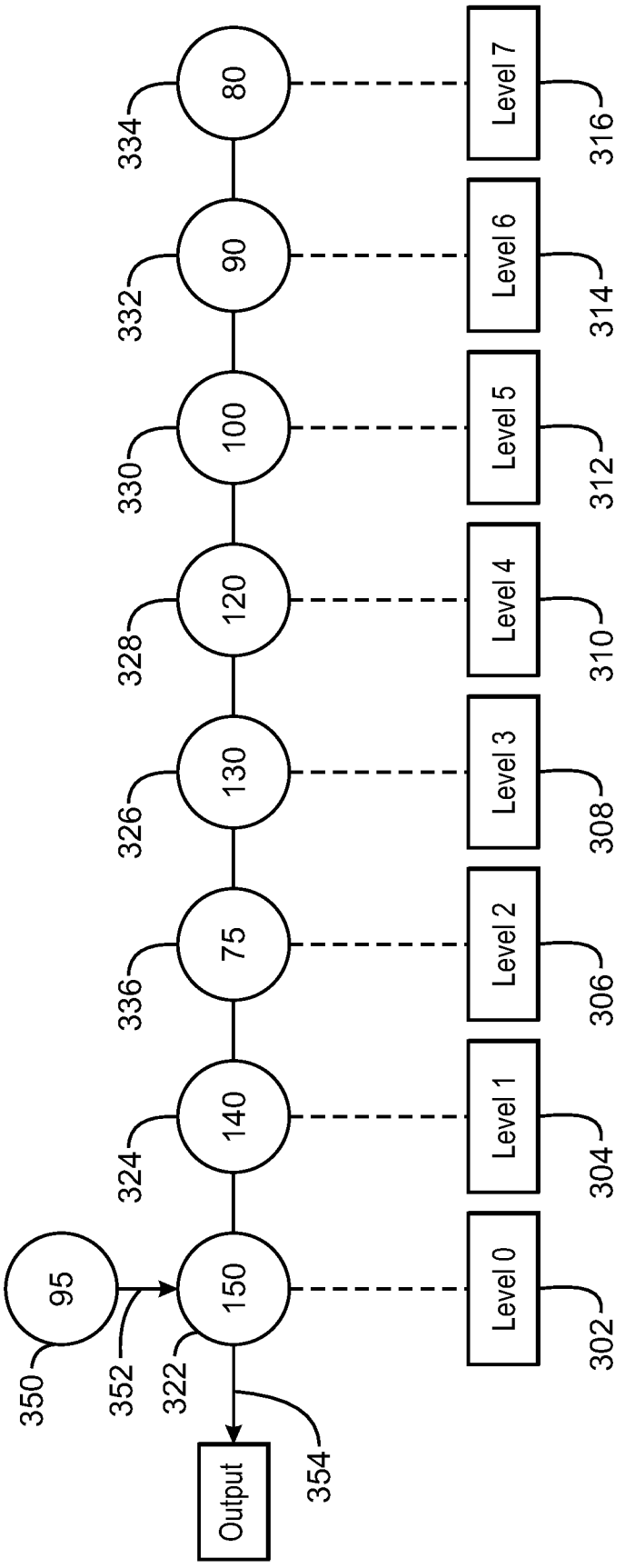
300C

FIG. 3C


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

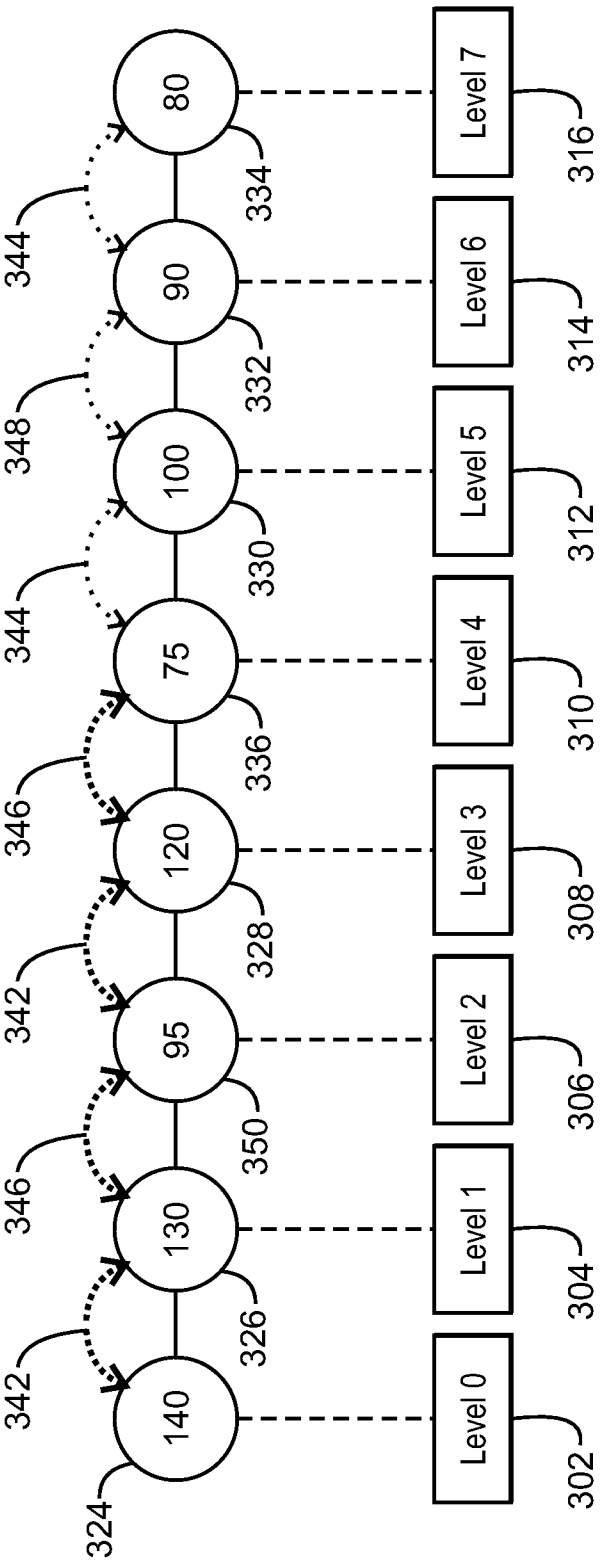
300D

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
95
150
140
75
130
120
100
90
80
324
336
326
328
330
332
334
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

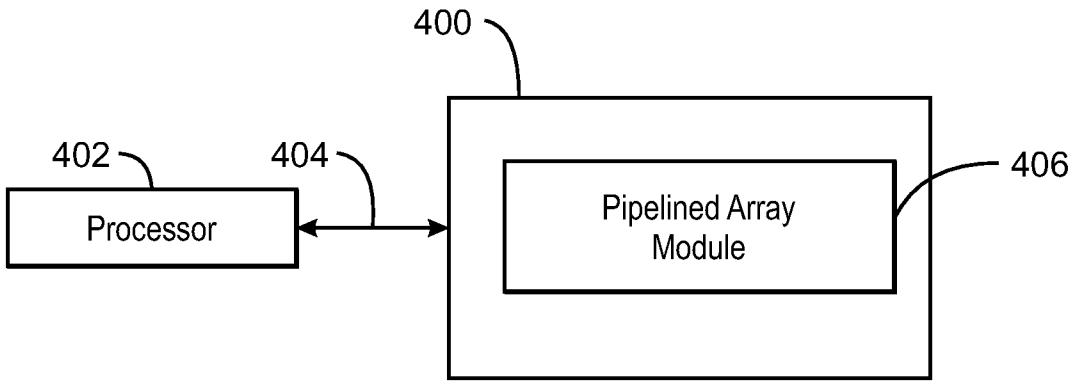


FIG. 4

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2014/049262

A. CLASSIFICATION OF SUBJECT MATTER

G06F 7/06(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 7/06; G06F 15/173; G06F 9/46; H04L 12/56

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: priority, data, entry, queue, swap, comparison, network, packet, quality of service

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	RANJITA BHAGWAN et al., `Fast and Scalable Priority Queue Architecture for High-Speed Network Switches`, In: INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings, IEEE, 2000, Vol. 2 See page 1, left column, lines 12-14; page 2, left column, lines 15-16, 27; page 2, right column, lines 17-18; page 6, right column, lines 8-11, 18-20; and figures 10, 11(a)-11(c).	1-2,5-7,9-10
Y		3-4,8,11-15
Y	US 2007-0208876 A1 (IAN EDWARD DAVIS) 06 September 2007 See paragraphs [0008], [0032], [0093]; and figure 5.	3-4,8,11-15
A	US 6778546 B1 (GARRY P. EPPS et al.) 17 August 2004 See column 26, lines 1-7; and figure 3.	1-15
A	US 2005-0289551 A1 (WALDEMAR WOJTKIEWICZ et al.) 29 December 2005 See paragraph [0036]; and figure 6.	1-15
A	US 2011-0252428 A1 (NAOTAKA MARUYAMA) 13 October 2011 See paragraph [0079]; and figure 13.	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
12 March 2015 (12.03.2015)	**12 March 2015 (12.03.2015)**

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
BYUN, Sung Cheal
Telephone No. +82-42-481-8262

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2014/049262

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2007-0208876 A1	06/09/2007	US 2012-0155466 A1	21/06/2012
		US 8671219 B2	11/03/2014
US 6778546 B1	17/08/2004	None	
US 2005-0289551 A1	29/12/2005	None	
US 2011-0252428 A1	13/10/2011	AU 2007-245508 A1	08/11/2007
		CN 101796487 A	04/08/2010
		CN 101796487 B	09/10/2013
		EP 2012971 A1	14/01/2009
		EP 2012971 B1	12/01/2011
		JP 4088335 B1	21/05/2008
		TW 200907794 A	16/02/2009
		TW 201421357 A	01/06/2014
		TW 201432562 A	16/08/2014
		TW I438679 B	21/05/2014
		US 2012-0064805 A1	15/03/2012
		US 2012-096472 A1	19/04/2012
		WO 2007-125250 A1	08/11/2007
		WO 2009-022369 A1	19/02/2009