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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
26 November 2020 (26.11.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/236274 A1

(51) International Patent Classification:
H04L 12/875 (2013.01) *H04L 12/861* (2013.01)
H04L 12/863 (2013.01) *H04L 12/24* (2006.01)

(21) International Application Number:
PCT/US2020/024248

(22) International Filing Date:
23 March 2020 (23.03.2020)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

62/852,273	23 May 2019 (23.05.2019)	US
62/852,203	23 May 2019 (23.05.2019)	US
62/852,289	23 May 2019 (23.05.2019)	US

(71) Applicant: **CRAY INC.** [US/US]; 901 5th Ave., STE 1000, Seattle, Washington 98164-2008 (US).

(72) Inventors: **ROWETH, Duncan**; 901 5th Ave., STE 1000, Seattle, Washington 98164 (US). **FROESE, Edwin L.**; 901 5th Ave., STE 1000, Seattle, Washington 98164 (US).

(74) Agent: **FEBBO, Michael** et al.; HEWLETT PACKARD ENTERPRISE, 3404 E. Harmony Road Mail Stop 79, 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, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IR, IS, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

WO 2020/236274 A1

(54) Title: SYSTEM AND METHOD FOR FACILITATING EFFICIENT EVENT NOTIFICATION MANAGEMENT FOR A NETWORK INTERFACE CONTROLLER (NIC)

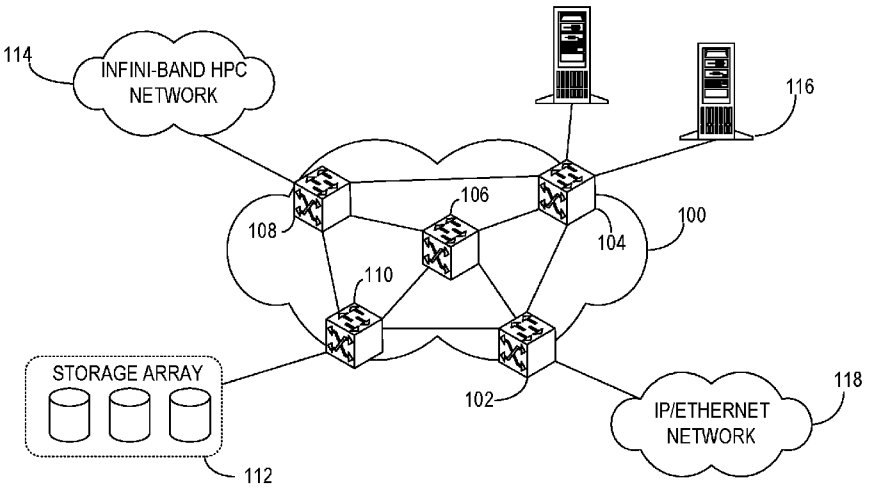

FIG. 1

(57) Abstract: A network interface controller (NIC) capable of efficient event management is provided. The NIC can be equipped with a host interface, a first memory device, and an event management module. During operation, the host interface can couple the NIC to a host device. The event management module can identify an event associated with an event queue stored in a second memory device of the host device. The event management module can insert, into a buffer, an event notification associated with the event. The buffer can be associated with the event queue and stored in the first memory device. If the buffer has met a release criterion, the event management module can insert, via the host interface, the aggregated event notifications into the event queue.

[Continued on next page]

(84) **Designated States** *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, 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))*
— *as to the applicant's entitlement to claim the priority of the earlier application (Rule 4.17(iii))*

Published:

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

SYSTEM AND METHOD FOR FACILITATING EFFICIENT EVENT NOTIFICATION MANAGEMENT FOR A NETWORK INTERFACE CONTROLLER (NIC)

Inventors: Duncan Roweth and Edwin L. Froese

BACKGROUND

Field

[0001] This is generally related to the technical field of networking. More specifically, this disclosure is related to systems and methods for facilitating a network interface controller (NIC) with efficient event management.

Related Art

[0002] As network-enabled devices and applications become progressively more ubiquitous, various types of traffic as well as the ever-increasing network load continue to demand more performance from the underlying network architecture. For example, applications such as high-performance computing (HPC), media streaming, and Internet of Things (IOT) can generate different types of traffic with distinctive characteristics. As a result, in addition to conventional network performance metrics such as bandwidth and delay, network architects continue to face challenges such as scalability, versatility, and efficiency.

SUMMARY

[0003] A network interface controller (NIC) capable of efficient event management is provided. The NIC can be equipped with a host interface, a first memory device, and an event management logic block. During operation, the host interface can couple the NIC to a host device. The event management logic block can identify an event associated with an event queue stored in a second memory device of the host device. The event management logic block can insert, into a buffer, an event notification associated with the event. The buffer can be associated with the event queue and stored in the first memory device. If the buffer has met a release criterion, the event management logic block can insert, via the host interface, the aggregated event notifications into the event queue.

BRIEF DESCRIPTION OF THE FIGURES

[0004] FIG. 1 shows an exemplary network.

[0005] FIG. 2A shows an exemplary NIC chip with a plurality of NICs.

[0006] FIG. 2B shows an exemplary architecture of a NIC.

[0007] FIG. 3A shows an exemplary efficient notification management process in a NIC.

[0008] FIG. 3B shows an exemplary combining buffer for facilitating efficient event notification management in a NIC.

[0009] FIG. 4A shows a flow chart of an event notification combination process in a NIC.

[0010] FIG. 4B shows a flow chart of an insertion process for a combining buffer in a NIC.

[0011] FIG. 4C shows a flow chart of a timer management process for a combining buffer of a NIC.

[0012] FIG. 5 shows an exemplary computer system equipped with a NIC that facilitates efficient event notification management.

[0013] In the figures, like reference numerals refer to the same figure elements.

DETAILED DESCRIPTION

[0014] Various modifications to the disclosed embodiments will be readily apparent to those skilled in the art, and the general principles defined herein may be applied to other embodiments and applications without departing from the spirit and scope of the present disclosure. Thus, the present invention is not limited to the embodiments shown.

Overview

[0015] The present disclosure describes systems and methods that facilitate efficient notification management in a network interface controller (NIC). The NIC allows a host to communicate with a data-driven network.

[0016] The embodiments described herein solve the problem of under-utilization of host interface width caused by inefficient event notification transfers by (i) aggregating event notifications in a combining buffer of a NIC, and (ii) providing the aggregated event notifications via the host interface, thereby utilizing the width of the host interface for a respective event notification.

[0017] During operation, an application, which can operate on a host device of a NIC, can issue a command for a data operation (e.g., a "GET" or a "PUT" command of remote direct memory access (RDMA)) to the NIC. Consequently, the host device can transfer the command (e.g., a direct memory access (DMA) descriptor of the command) to the NIC. Upon completion

of the operation, the NIC may notify the host device that the operation has been completed. In addition to the event completion notifications, the NIC may also need to provide other event notifications, such as an error notification or telemetry data obtained via measurements. Typically, the memory of the host device can maintain one or more event queues associated with different events. The NIC can send an event notification that writes into a corresponding event queue

[0018] Upon receiving the event notification via the host interface, the host device can write the event notification into the event queue of the memory. The application can obtain the event notification from the event queue. However, an event notification may be relatively short compared to the bit width (or width) of the host interface. For example, the host interface can be a peripheral component interconnect express (PCIe) interface that can support N bytes of data transfer per clock cycle. In other words, the width of the host interface can be N bytes. However, a typical event notification may be M, where $M < N$, bytes long. Consequently, issuing PCIe write operation for an M-byte event notification can cause significant underutilization of the host interface.

[0019] To solve this problem, the NIC can aggregate a plurality of event notifications into a single notification and send the aggregated event notification via the host interface. The aggregated notification may have a length that can correspond to the width of the interface. As a result, the aggregated notification can efficiently utilize the width of the interface. During operation, the NIC can generate an event notification for an event and determine a corresponding event queue associated with the event. The NIC can then allocate the event notification to a combining buffer associated with the event queue. The NIC can continue to aggregate notifications associated with the event until a new event notification cannot be included in the buffer, or the new event notification fills the buffer. The NIC can then issue an interface-based write operation that inserts the event notifications from the buffer to the corresponding event queue. In this way, the NIC can utilize the width of the host interface, thereby facilitating efficient event management for the NIC.

[0020] One embodiment of the present invention provides a NIC that can be equipped with a host interface, a first memory device, and an event management logic block. During operation, the host interface can couple the NIC to a host device. The event management logic block can identify an event associated with an event queue stored in a second memory device of the host device. The event management logic block can insert, into a buffer, an event notification associated with the event. The buffer can be associated with the event queue and stored in the first memory device. If the buffer has met a release criterion, the event management logic block can insert, via the host interface, the aggregated event notifications into the event queue.

[0021] In a variation on this embodiment, the event management logic block can determine that the buffer has met the release criterion by determining that a timer for the buffer has reached a latency tolerance for the event queue.

[0022] In a further variation, the second memory device may store a plurality of event queues. A respective event queue can then be associated with a distinct latency tolerance.

[0023] In a variation on this embodiment, the buffer can be distributed across a plurality of memory logic blocks of the first memory device.

[0024] In a further variation, the granularity of the event notification may correspond to a multiple of a size of a memory logic block.

[0025] In a variation on this embodiment, if a new buffer has not been allocated for the event queue, the event management logic block can allocate the new buffer for the event queue.

[0026] In a variation on this embodiment, the release criterion can be based on one or more of: (i) the buffer not having the capacity for a new event notification, and (ii) the buffer being filled by the insertion of the event notification.

[0027] In a variation on this embodiment, if the event notification is an initial notification, the event management logic block can insert null events into a remainder of the buffer.

[0028] In a variation on this embodiment, the event management logic block can store event notifications of different sizes in the same buffer.

[0029] In a variation on this embodiment, the host interface can be a peripheral component interconnect express (PCIe) interface. The event management logic block can then insert the aggregated event notifications into the event queue based on a PCIe write.

[0030] In this disclosure, the description in conjunction with FIG. 1 is associated with the network architecture and the description in conjunction with FIG. 2A and onward provide more details on the architecture and operations associated with a NIC that supports efficient event management.

[0031] FIG. 1 shows an exemplary network. In this example, a network 100 of switches, which can also be referred to as a "switch fabric," can include switches 102, 104, 106, 108, and 110. Each switch can have a unique address or ID within switch fabric 100. Various types of devices and networks can be coupled to a switch fabric. For example, a storage array 112 can be coupled to switch fabric 100 via switch 110; an InfiniBand (IB) based HPC network 114 can be coupled to switch fabric 100 via switch 108; a number of end hosts, such as host 116, can be coupled to switch fabric 100 via switch 104; and an IP/Ethernet network 118 can be coupled to switch fabric 100 via switch 102. In general, a switch can have edge ports and fabric ports. An edge port can couple to a device that is external to the fabric. A fabric port can couple to another

switch within the fabric via a fabric link. Typically, traffic can be injected into switch fabric 100 via an ingress port of an edge switch, and leave switch fabric 100 via an egress port of another (or the same) edge switch. An ingress link can couple a NIC of an edge device (for example, an HPC end host) to an ingress edge port of an edge switch. Switch fabric 100 can then transport the traffic to an egress edge switch, which in turn can deliver the traffic to a destination edge device via another NIC.

Exemplary NIC Architecture

[0032] FIG. 2A shows an exemplary NIC chip with a plurality of NICs. With reference to the example in FIG. 1, a NIC chip 200 can be a custom application-specific integrated circuit (ASIC) designed for host 116 to work with switch fabric 100. In this example, chip 200 can provide two independent NICs 202 and 204. A respective NIC of chip 200 can be equipped with a host interface (HI) (e.g., an interface for connecting to the host processor) and one High-speed Network Interface (HNI) for communicating with a link coupled to switch fabric 100 of FIG. 1. For example, NIC 202 can include an HI 210 and an HNI 220, and NIC 204 can include an HI 211 and an HNI 221.

[0033] In some embodiments, HI 210 can be a peripheral component interconnect (PCI) or a peripheral component interconnect express (PCIe) interface. HI 210 can be coupled to a host via a host connection 201, which can include *N* (e.g., *N* can be 16 in some chips) PCIe Gen 4 lanes capable of operating at signaling rates up to 25 Gbps per lane. HNI 210 can facilitate a high-speed network connection 203, which can communicate with a link in switch fabric 100 of FIG. 1. HNI 210 can operate at aggregate rates of either 100 Gbps or 200 Gbps using M (e.g., *M* can be 4 in some chips) full-duplex serial lanes. Each of the M lanes can operate at 25 Gbps or 50 Gbps based on non-return-to-zero (NRZ) modulation or pulse amplitude modulation 4 (PAM4), respectively. HNI 220 can support the Institute of Electrical and Electronics Engineers (IEEE) 802.3 Ethernet-based protocols as well as an enhanced frame format that provides support for higher rates of small messages.

[0034] NIC 202 can support one or more of: point-to-point message passing based on Message Passing Interface (MPI), remote memory access (RMA) operations, offloading and progression of bulk data collective operations, and Ethernet packet processing. When the host issues an MPI message, NIC 202 can match the corresponding message type. Furthermore, NIC 202 can implement both eager protocol and rendezvous protocol for MPI, thereby offloading the corresponding operations from the host.

[0035] Furthermore, the RMA operations supported by NIC 202 can include PUT, GET, and Atomic Memory Operations (AMO). NIC 202 can provide reliable transport. For example,

if NIC 202 is a source NIC, NIC 202 can provide a retry mechanism for idempotent operations. Furthermore, connection-based error detection and retry mechanism can be used for ordered operations that may manipulate a target state. The hardware of NIC 202 can maintain the state necessary for the retry mechanism. In this way, NIC 202 can remove the burden from the host (e.g., the software). The policy that dictates the retry mechanism can be specified by the host via the driver software, thereby ensuring flexibility in NIC 202.

[0036] Furthermore, NIC 202 can facilitate triggered operations, a general-purpose mechanism for offloading, and progression of dependent sequences of operations, such as bulk data collectives. NIC 202 can support an application programming interface (API) (e.g., libfabric API) that facilitates fabric communication services provided by switch fabric 100 of FIG. 1 to applications running on host 116. NIC 202 can also support a low-level network programming interface, such as Portals API. In addition, NIC 202 can provide efficient Ethernet packet processing, which can include efficient transmission if NIC 202 is a sender, flow steering if NIC 202 is a target, and checksum computation. Moreover, NIC 202 can support virtualization (e.g., using containers or virtual machines).

[0037] FIG. 2B shows an exemplary architecture of a NIC. In NIC 202, the port macro of HNI 220 can facilitate low-level Ethernet operations, such as physical coding sublayer (PCS) and media access control (MAC). In addition, NIC 202 can provide support for link layer retry (LLR). Incoming packets can be parsed by parser 228 and stored in buffer 229. Buffer 229 can be a PFC Buffer provisioned to buffer a threshold amount (e.g., one microsecond) of delay bandwidth. HNI 220 can also include control transmission unit 224 and control reception unit 226 for managing outgoing and incoming packets, respectively.

[0038] NIC 202 can include a Command Queue (CQ) unit 230. CQ unit 230 can be responsible for fetching and issuing host side commands. CQ unit 230 can include command queues 232 and schedulers 234. Command queues 232 can include two independent sets of queues for initiator commands (PUT, GET, etc.) and target commands (Append, Search, etc.), respectively. Command queues 232 can be implemented as circular buffers maintained in the memory of NIC 202. Applications running on the host can write to command queues 232 directly. Schedulers 234 can include two separate schedulers for initiator commands and target commands, respectively. The initiator commands are sorted into flow queues 236 based on a hash function. One of flow queues 236 can be allocated to a unique flow. Furthermore, CQ unit 230 can further include a triggered operations module (or logic block) 238, which is responsible for queuing and dispatching triggered commands.

[0039] Outbound transfer engine (OXE) 240 can pull commands from flow queues 236 in order to process them for dispatch. OXE 240 can include an address translation request unit

(ATRU) 244 that can send address translation requests to address translation unit (ATU) 212. ATU 212 can provide virtual to physical address translation on behalf of different engines, such as OXE 240, inbound transfer engine (IXE) 250, and event engine (EE) 216. ATU 212 can maintain a large translation cache 214. ATU 212 can either perform translation itself or may use host-based address translation services (ATS). OXE 240 can also include message chopping unit (MCU) 246, which can fragment a large message into packets of sizes corresponding to a maximum transmission unit (MTU). MCU 246 can include a plurality of MCU modules. When an MCU module becomes available, the MCU module can obtain the next command from an assigned flow queue. The received data can be written into data buffer 242. The MCU module can then send the packet header, the corresponding traffic class, and the packet size to traffic shaper 248. Shaper 248 can determine which requests presented by MCU 246 can proceed to the network.

[0040] Subsequently, the selected packet can be sent to packet and connection tracking (PCT) 270. PCT 270 can store the packet in a queue 274. PCT 270 can also maintain state information for outbound commands and update the state information as responses are returned. PCT 270 can also maintain packet state information (e.g., allowing responses to be matched to requests), message state information (e.g., tracking the progress of multi-packet messages), initiator completion state information, and retry state information (e.g., maintaining the information required to retry a command if a request or response is lost). If a response is not returned within a threshold time, the corresponding command can be retrieved from retry buffer 272. PCT 270 can facilitate connection management for initiator and target commands based on source tables 276 and target tables 278, respectively. For example, PCT 270 can update its source tables 276 to track the necessary state for reliable delivery of the packet and message completion notification. PCT 270 can forward outgoing packets to HNI 220, which stores the packets in outbound queue 222.

[0041] NIC 202 can also include an IXE 250, which provides packet processing if NIC 202 is a target or a destination. IXE 250 can obtain the incoming packets from HNI 220. Parser 256 can parse the incoming packets and pass the corresponding packet information to a List Processing Engine (LPE) 264 or a Message State Table (MST) 266 for matching. LPE 264 can match incoming messages to buffers. LPE 264 can determine the buffer and start address to be used by each message. LPE 264 can also manage a pool of list entries 262 used to represent buffers and unexpected messages. MST 266 can store matching results and the information required to generate target side completion events. An event can be an internal control message for communication among the elements of NIC 202. MST 266 can be used by unrestricted

operations, including multi-packet PUT commands, and single-packet and multi-packet GET commands.

[0042] Subsequently, parser 256 can store the packets in packet buffer 254. IXE 250 can obtain the results of the matching for conflict checking. DMA write and AMO module 252 can then issue updates to the memory generated by write and AMO operations. If a packet includes a command that generates target side memory read operations (e.g., a GET request), the packet can be passed to the OXE 240. NIC 202 can also include an EE 216, which can receive requests to generate event notifications from other modules or units in NIC 202. An event notification can specify that either a full event or a counting event is generated. EE 216 can manage event queues, located within host processor memory, to which it writes full events. EE 216 can forward counting events to CQ unit 230.

Event Management in NIC

[0043] FIG. 3A shows an exemplary efficient notification management process in a NIC. In this example, a host device 300 can be equipped with a NIC 330. Device 300 can include a processor 302, a memory device 304, and an interface system 306. An HI 332 of NIC 330 can be coupled to interface system 306 of device 300. In some embodiments, HI 332 can be a PCIe interface, and interface system 306 can be a PCIe system that provides a slot for HI 332. NIC 330 can also include an EE 334 for managing events, as described in conjunction with FIG. 2B.

[0044] Typically, an application 308 running on device 300 can issue a command for a data operation (e.g., an RDMA operation) to NIC 330. Consequently, device 300 can transfer the command to NIC 330. Upon completion of the operation, NIC 330 may notify device 330 that the operation has been completed. In addition to the event completion notifications, NIC 330 may also need to provide other event notifications, such as an error notification or telemetry data obtained via measurements, to device 300. Typically, memory device 304 can maintain one or more event queues associated with different events. For example, memory device 304 can include an event queue 312 for a type of event, such as notifications for write operations.

[0045] Upon completion of an event, EE 334 can generate an event notification 322 (e.g., a DMA descriptor) associated with an event. EE 334 can then issue a corresponding interface-based write operation for storing event notification 322 into event queue 312. The write operation can be associated with an enqueue operation into event queue 312. Upon receiving event notification 322 via interface system 306, processor 302 can write event notification 322 into event queue 312. Application 308 can obtain event notification 322 from event queue 312. In this way, application 308 can determine whether an operation issued from application 308 has been completed.

[0046] However, event notification 322 may be relatively short compared to width 360 of interface system 306. For example, interface system 306 can provide a PCIe interface that can support 64 bytes of data transfer per clock cycle. Hence, width 360 of interface system 306 can be 64 bytes. However, event notification 322 may be significantly shorter than width 360. Consequently, issuing PCIe write operation (e.g., the enqueue operation into event queue 312) for event notification 322 can cause significant underutilization of the transfer capability offered by width 360.

[0047] To solve this problem, NIC 330 can aggregate a plurality of event notifications associated with event queue 312 into a single notification 320 and send aggregated event notification 320 via HI 332. During operation, upon retrieving an event, EE 334 can generate an event notification 322 and determine that event queue 312 is associated with the event. Instead of sending event notification 322 to host 300, EE 334 can identify a combining buffer 314 in NIC 330. Here, combining buffer 314 can be allocated for aggregating event notifications destined to event queue 312. If no combining buffer is allocated for event queue 312 when event notification 322 is generated, NIC 330 can allocate a new combining buffer (i.e., combining buffer 314 can be newly allocated buffer). EE 334 can then store event notification 322 in combining buffer 314.

[0048] Similarly, upon generating another event notification 324 for event queue 312, EE 334 can determine whether a combining buffer has been allocated for event queue 312. EE 334 can identify combining buffer 314 and insert event notification 324 into combining buffer 314. At this point, combining buffer 314 can store event notifications 322 and 324. In this way, EE 334 can continue to aggregate notifications associated with event queue 312. EE 334 can then determine whether combining buffer 314 meets a release criterion. For example, if a subsequent event notification cannot be included in combining buffer 314 or event notification 324 fills combining buffer 314, EE 334 can determine that combining buffer 314 has met the release criterion and should be inserted into event queue 312. Accordingly, EE 334 can initiate data transfer to host 300.

[0049] EE 334 can aggregate the content of combining buffer 314, which can include event notifications 322 and 324, into a single aggregated notification 320. Aggregated notification 320 may have a length that can correspond to width 360. If the combined length of event notifications 322 and 324 is smaller than width 360, EE 334 may pad aggregated notification 320 with null events (e.g., null values) to adjust to width 360. For efficient padding, NIC 330 may pad combining buffer 314 with the null events upon inserting event notification 322. Upon insertion into combining buffer 314, event notification 324 can replace the null event subsequent to event notification 322 in combining buffer 314. EE 334 can then issue an

interface-based write operation that inserts aggregated notification 320 to event queue 312. In this way, aggregated notification 320 can efficiently utilize width 360, thereby facilitating efficient event management for NIC 330.

[0050] In some embodiments, event queue 312 can be configured with a latency tolerance. Combining buffer 314 can store event notifications for a duration indicated by the latency tolerance. If the latency tolerance is set to zero, upon receiving event notification 322 and inserting into combining buffer 314, EE 334 can insert the content of combining buffer 314 into event queue 312. Since the rest of combining buffer 314 can be padded with null events, EE 334 can readily issue the write operation via HI 332. However, if the latency tolerance is set to a non-zero value, EE 334 may wait for more event notifications until a timer corresponding to the latency tolerance expires. Suppose that device 300 maintains a plurality of event queues in memory device 304. Two different event queues can then have the same or two distinct latency tolerances.

[0051] FIG. 3B shows an exemplary combining buffer for facilitating efficient event notification management in a NIC. Combining buffer 314 can be facilitated based on a number of buffer modules 352, 354, 356, and 358. In some embodiments, a buffer module can be a random access memory (RAM) device. For example, if width 360 in FIG. 3A is 64 bytes, combining buffer 314 may also have the capacity to hold 64 bytes. Consequently, each of buffer modules 352, 354, 356, and 358 can store at least 16 bytes of data. In this way, four separate buffer modules can facilitate the overall width of 64 bytes. Furthermore, each of buffer modules 352, 354, 356, and 358 can include additional capacity to store error-correcting codes (e.g., a single error correction/double error detection (SECDED) code). The SECDED code stored in a buffer module can be used to protect that buffer module.

[0052] Combining multiple buffer modules to form combining buffer 314 can allow individual event notifications (e.g., individual 16-byte segments) to be written while leaving other segments unmodified. As a result, if buffer module 352 already stores event notification 322, inserting another event notification 324 into buffer modules 356 and 358 may not affect the content of buffer module 352. As a result, inserting an event notification into partially-filled combining buffer 314 may not need reading the content of combining buffer 314, modifying the content, and rewriting the modified content into combining buffer 314. Since a buffer module may support a single read and write per clock cycle, one of the buffer modules can be read from while another of the buffer modules can be written into in the same clock cycle.

[0053] In some embodiments, an event notification can be generated to align with the size of a buffer module. Therefore, the size of an event notification can be a multiple of the size of a buffer module. For example, if the size of a buffer module is 16 bytes, the size of an event

notification can be 16, 32, or 64 bytes long, depending on the content of the event notification. EE 334 may store event notifications of different sizes in the same combining buffer 314. An event notification can be stored at an address offset aligned for the size of the event notification. For example, a 16-byte event notification can be written at a location aligned with a 16 byte-aligned address. If an event notification does not align with the size of a buffer module, EE 334 may insert null events to provide the alignment.

[0054] If the size of a buffer module is 16 bytes, in this example, event notifications 322 and 324 can be 32 and 16 bytes long, respectively. Consequently, event notification 322 can be stored in buffer modules 352 and 354, and event notification 324 can be stored in buffer module 356. If EE 334 determines that the subsequent event notification does not fit in buffer module 358, EE 334 can enqueue the content of combining buffer 314. When EE 334 writes the first event notification in combining buffer 314, EE 334 can simultaneously write null events into the reminder of combining buffer 314. For example, upon writing event notification 322 in buffer modules 352 354, EE 334 can write null events into buffer modules 356 and 358. This allows EE 334 to initiate the enqueue operation without the need to write null events to any available capacity that may exist at that time.

[0055] FIG. 4A shows a flow chart of an event notification combination process in a NIC. During operation, an EE of the NIC can generate an event notification associated with an event queue (operation 402). The EE can determine whether a combining buffer is allocated to the event queue (operation 404). If a combining buffer is not allocated to the event queue, the EE can allocate a combining buffer for the event queue and insert the event notification into the combining buffer (operation 414). If a combining buffer is allocated to the event queue, the EE can determine whether the combining buffer has available capacity for event notification (operation 406).

[0056] If the combining buffer does not have sufficient available capacity, the EE can enqueue the combining buffer, allocate a new combining buffer for the event queue, and insert the event notification into the new combining buffer (operation 416). On the other hand, if the combining buffer has available capacity, the EE can insert the event notification into the combining buffer (operation 408). Upon inserting the event notification (operation 408, 414, or 416), the EE can determine whether the combining buffer is full (e.g., due to the insertion) (operation 410). If the combining buffer is full, the EE can enqueue the combining buffer (operation 412). If the combining buffer is not full (operation 410) or upon performing the enqueue operation (operation 412), the EE can continue to generate another event notification associated with an event queue (operation 402).

[0057] FIG. 4B shows a flow chart of an insertion process for a combining buffer in a NIC. During operation, an EE of the NIC can identify a combining buffer for an event notification (operation 432). The EE can then determine a location for insertion in the combining buffer (operation 434) and insert the event notification into the combining buffer at the determined location (operation 436). Subsequently, the EE can determine whether the insertion is the initial insertion (operation 438). If the insertion is the initial insertion, the EE can insert null events into the rest of the combining buffer (operation 440).

[0058] FIG. 4C shows a flow chart of a timer management process for a combining buffer of a NIC. During operation, the EE of the NIC can determine state information associated with a respective combining buffer in parallel (operation 452). The NIC can maintain a combining buffer for a respective event queue. Since the host device of the NIC can include a plurality of event queues, the NIC can maintain a plurality of combining buffers. The EE can then calculate a current time based on a timer counter (operation 454).

[0059] Subsequently, the EE can compare the state of a respective combining buffer with the current time and a tolerance value to identify combining buffers that have expired (operation 456). The EE can set the state of a respective identified combining buffer as expired and enqueue the combining buffer (operation 458). The EE can then increment the timer counter (operation 460). In some embodiments, the EE may increment the timer counter at a predetermined interval. Upon each increment, the EE can check which combining buffers have expired.

Exemplary Computer System

[0060] FIG. 5 shows an exemplary computer system equipped with a NIC that facilitates efficient event notification management. Computer system 550 includes a processor 552, a memory device 554, and a storage device 556. Memory device 554 can include a volatile memory device (e.g., a dual in-line memory module (DIMM)). Furthermore, computer system 550 can be coupled to a keyboard 562, a pointing device 564, and a display device 566. Storage device 556 can store an operating system 570. An application 572 can operate on operating system 570.

[0061] Computer system 550 can be equipped with a host interface coupling a NIC 520 that facilitates efficient event management. NIC 520 can provide one or more HNIs to computer system 550. NIC 520 can be coupled to a switch 502 via one of the HNIs. NIC 520 can include an event logic block 530, as described in conjunction with FIGs. 2B and 3. Event logic block 530 can include a notification logic block 532 and an enqueue logic block 534. Event logic

block 530 can maintain a combining buffer 536, which can be associated with an event queue 560 in memory device 554.

[0062] Notification logic block 532 can generate an event notification and store the event notification in combining buffer 536. Notification logic block 532 can determine a release criterion for combining 536. The release criterion can indicate one or more of: whether combining buffer 536 has sufficient capacity for the next event notification and whether combining buffe 536 is full, and whether an event notification has been held in combining buffer 536, when it is partially full, for a sufficiently long duration so as to have reached the latency tolerance of event queue 560. If combining buffer 536 has met the release criterion (e.g., has stored a sufficient number of event notifications), enqueue logic block 534 can enqueue the content of combining buffer 536 into event queue 560 (e.g., based on a write request via the HI).

[0063] In summary, the present disclosure describes a NIC that facilitates efficient event management. The NIC can be equipped with a host interface, a first memory device, and an event management logic block. During operation, the host interface can couple the NIC to a host device. The event management logic block can identify an event associated with an event queue stored in a second memory device of the host device. The event management logic block can insert, into a buffer, an event notification associated with the event. The buffer can be associated with the event queue and stored in the first memory device. If the buffer has met a release criterion, the event management logic block can insert, via the host interface, the aggregated event notifications into the event queue.

[0064] The methods and processes described above can be performed by hardware logic blocks, modules, or apparatus. The hardware logic blocks, modules, logic blocks, or apparatus can include, but are not limited to, application-specific integrated circuit (ASIC) chips, field-programmable gate arrays (FPGAs), dedicated or shared processors that execute a piece of code at a particular time, and other programmable-logic devices now known or later developed. When the hardware logic blocks, modules, or apparatus are activated, they perform the methods and processes included within them.

[0065] The methods and processes described herein can also be embodied as code or data, which can be stored in a storage device or computer-readable storage medium. When a processor reads and executes the stored code or data, the processor can perform these methods and processes.

[0066] The foregoing descriptions of embodiments of the present invention have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the present invention to the forms disclosed. Accordingly, many modifications and variations will be apparent to practitioners skilled in the art. Additionally, the

above disclosure is not intended to limit the present invention. The scope of the present invention is defined by the appended claims.

What Is Claimed Is:

1. A network interface controller (NIC), comprising:

a host interface coupling a host device;

a first memory device; and

an event management logic block to:

identify an event associated with an event queue stored in a second memory device of the host device;

insert, into a buffer stored in the first memory device, an event notification associated with the event, wherein the buffer is associated with the event queue; and

in response to determining that the buffer has met a release criterion, insert, via the host interface, the aggregated event notifications into the event queue.

2. The network interface controller of claim 1, wherein determining that the buffer has met the release criterion further comprises determining that a timer for the buffer has reached a latency tolerance for the event queue.

3. The network interface controller of claim 3, wherein the second memory device to store a plurality of event queues, and wherein a respective event queue is associated with a distinct latency tolerance.

4. The network interface controller of claim 1, wherein the buffer is distributed across a plurality of memory modules of the first memory device.

5. The network interface controller of claim 3, wherein the granularity of the event notification corresponds to a multiple of a size of a memory module.

6. The network interface controller of claim 1, wherein the event management logic block is further to, in response to a new buffer not being allocated for the event queue, allocate the new buffer for the event queue.

7. The network interface controller of claim 1, wherein the release criterion is based on one or more of:

the buffer not having capacity for a new event notification; and

the buffer being filled by the insertion of the event notification.

8. The network interface controller of claim 1, wherein the event management logic block is further to, in response to the event notification being an initial notification, insert null events into a reminder of the buffer.

9. The network interface controller of claim 1, wherein the event management logic block is to store event notifications of different sizes in the same buffer.

10. The network interface controller of claim 1, wherein the host interface is a peripheral component interconnect express (PCIe) interface; and

wherein the event management logic block is further insert the aggregated event notifications into the event queue based on a PCIe write.

11. A method for facilitating efficient event management in a network interface controller (NIC), the method comprising:

maintaining a buffer in a first memory device of the NIC;

identifying an event associated with an event queue stored in a second memory device of a host device of the NIC;

inserting, into the buffer, an event notification associated with the event, wherein the buffer is associated with the event queue; and

in response to determining that the buffer has met a release criterion, inserting, via a host interface coupling the host device, the aggregated event notifications into the event queue.

12. The method of claim 11, wherein determining that the buffer has met the release criterion further comprises determining that a timer for the buffer has reached a latency tolerance for the event queue.

13. The method of claim 13, wherein the second memory device to store a plurality of event queues, and wherein a respective event queue is associated with a distinct latency tolerance.

14. The method of claim 11, wherein the buffer is distributed across a plurality of memory modules of the first memory device.

15. The method of claim 13, wherein the granularity of the event notification corresponds to a multiple of a size of a memory module.

16. The method of claim 11, further comprising, in response to a new buffer not being allocated for the event queue, allocating the new buffer for the event queue.

17. The method of claim 11, wherein the release criterion is based on one or more of:

the buffer not having capacity for a new event notification; and

the buffer being filled by the insertion of the event notification.

18. The method of claim 11, further comprising, in response to the event notification being an initial notification, inserting null events into a reminder of the buffer.

19. The method of claim 11, further comprising storing event notifications of different sizes in the same buffer.

20. The method of claim 11, wherein the host interface is a peripheral component interconnect express (PCIe) interface; and

wherein the method further comprises inserting the aggregated event notifications into the event queue based on a PCIe write.

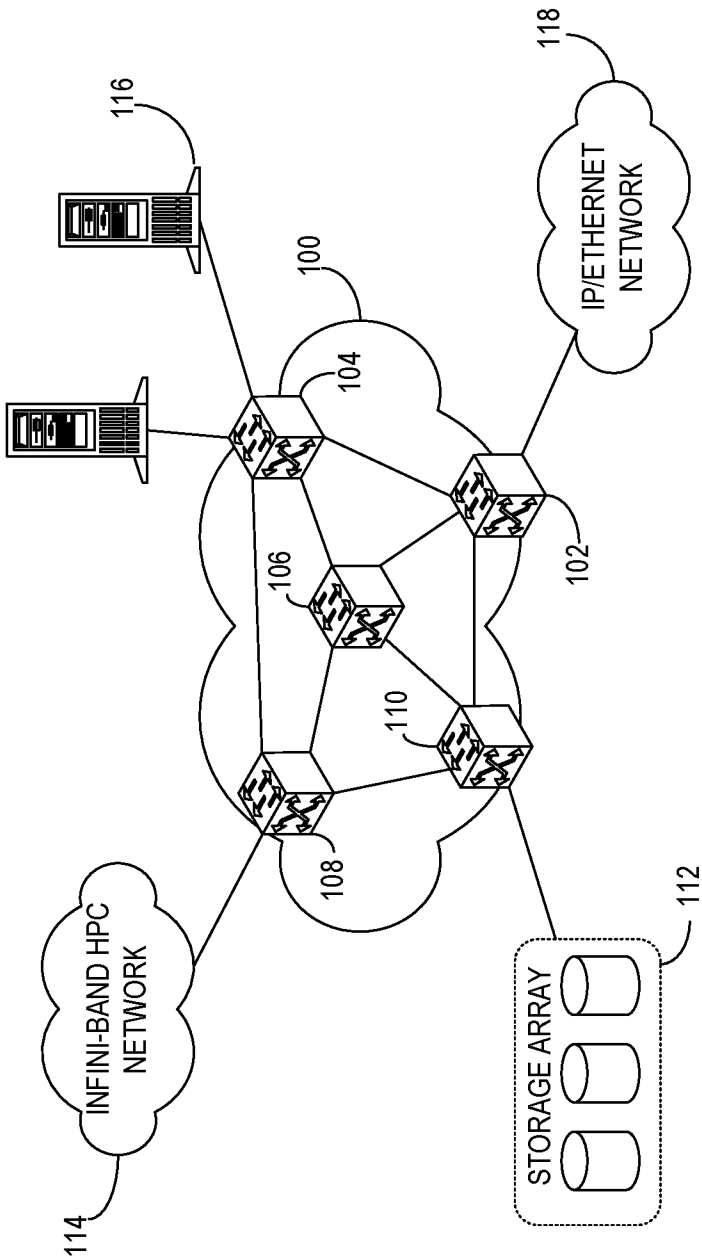
116
118
IP/ETHERNET NETWORK
100
104
102
106
110
108
INFINI-BAND HPC NETWORK
114
STORAGE ARRAY
112

FIG. 1

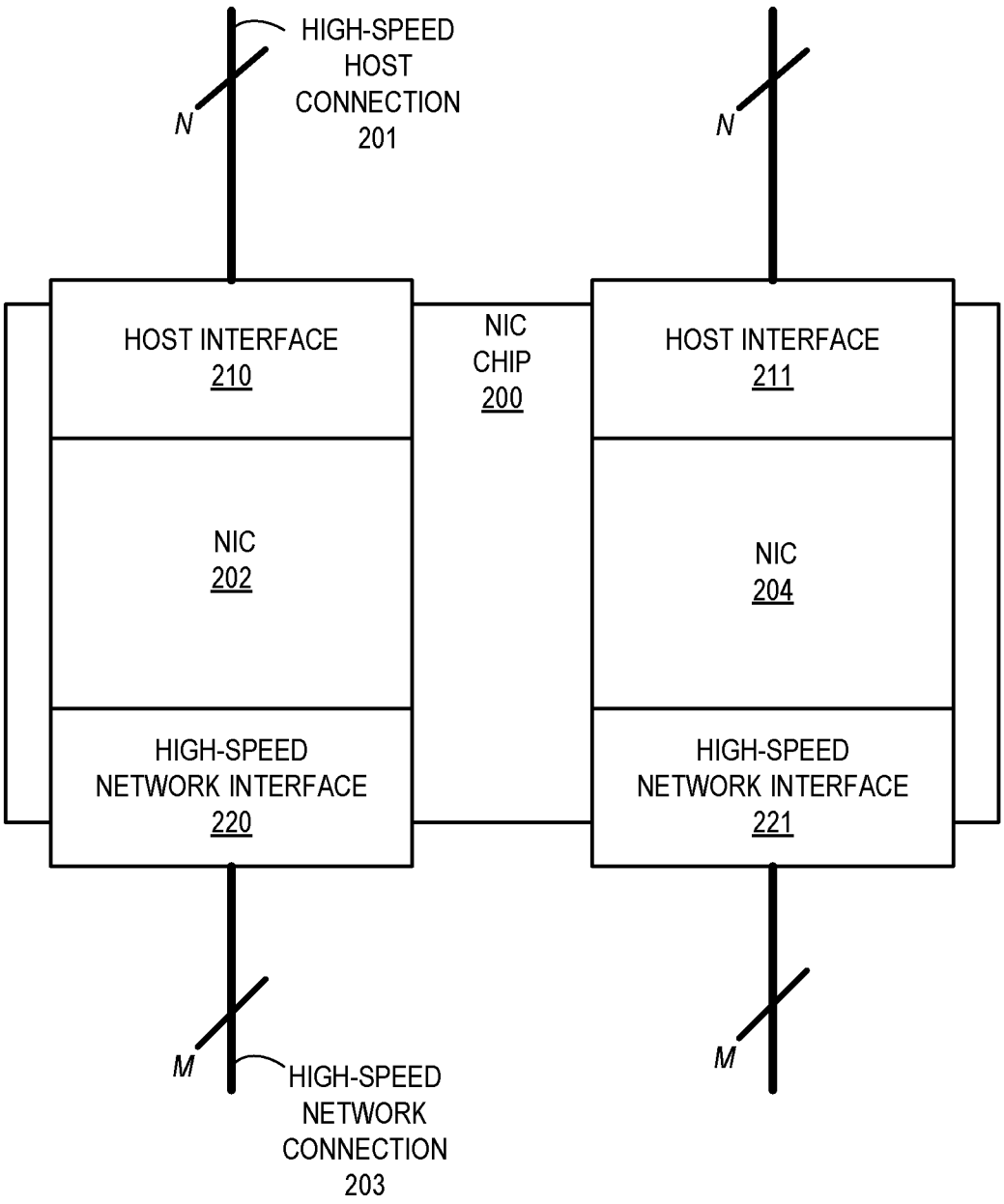
HIGH-SPEED
HOST
CONNECTION
201
N
N
HOST INTERFACE
210
NIC
CHIP
200
HOST INTERFACE
211
NIC
202
NIC
204
HIGH-SPEED
NETWORK INTERFACE
220
HIGH-SPEED
NETWORK INTERFACE
221
M
M
HIGH-SPEED
NETWORK
CONNECTION
203

FIG. 2A

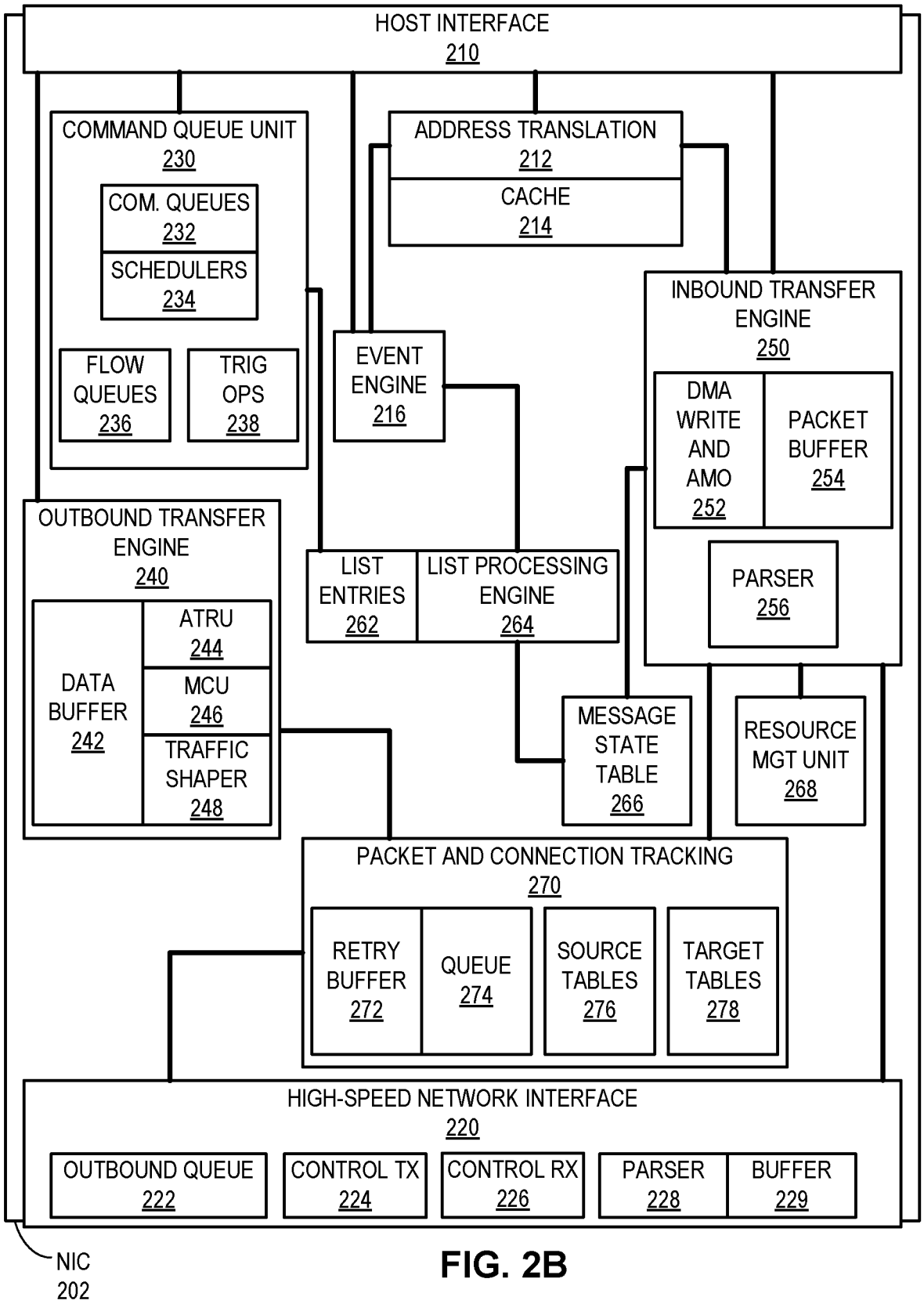
HOST INTERFACE
210
COMMAND QUEUE UNIT
230
COM. QUEUES
232
SCHEDULERS
234
FLOW QUEUES
236
TRIG OPS
238
ADDRESS TRANSLATION
212
CACHE
214
EVENT ENGINE
216
INBOUND TRANSFER ENGINE
250
DMA WRITE AND AMO
252
PACKET BUFFER
254
PARSER
256
OUTBOUND TRANSFER ENGINE
240
DATA BUFFER
242
ATRU
244
MCU
246
TRAFFIC SHAPER
248
LIST ENTRIES
262
LIST PROCESSING ENGINE
264
MESSAGE STATE TABLE
266
RESOURCE MGT UNIT
268
PACKET AND CONNECTION TRACKING
270
RETRY BUFFER
272
QUEUE
274
SOURCE TABLES
276
TARGET TABLES
278
HIGH-SPEED NETWORK INTERFACE
220
OUTBOUND QUEUE
222
CONTROL TX
224
CONTROL RX
226
PARSER
228
BUFFER
229
NIC
202

FIG. 2B

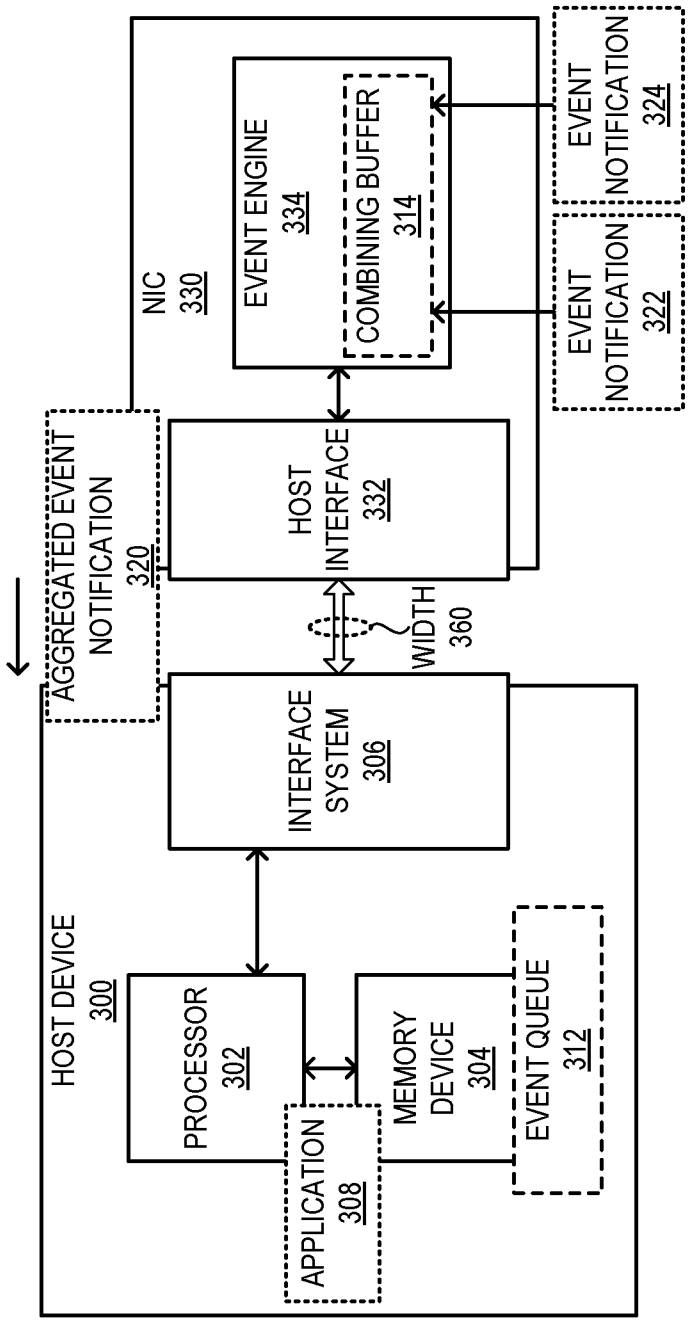
HOST DEVICE
300
PROCESSOR
302
APPLICATION
308
MEMORY
DEVICE
304
EVENT QUEUE
312
INTERFACE
SYSTEM
306
WIDTH
360
AGGREGATED EVENT
NOTIFICATION
320
HOST
INTERFACE
332
NIC
330
EVENT ENGINE
334
COMBINING BUFFER
314
EVENT
NOTIFICATION
322
EVENT
NOTIFICATION
324

FIG. 3A

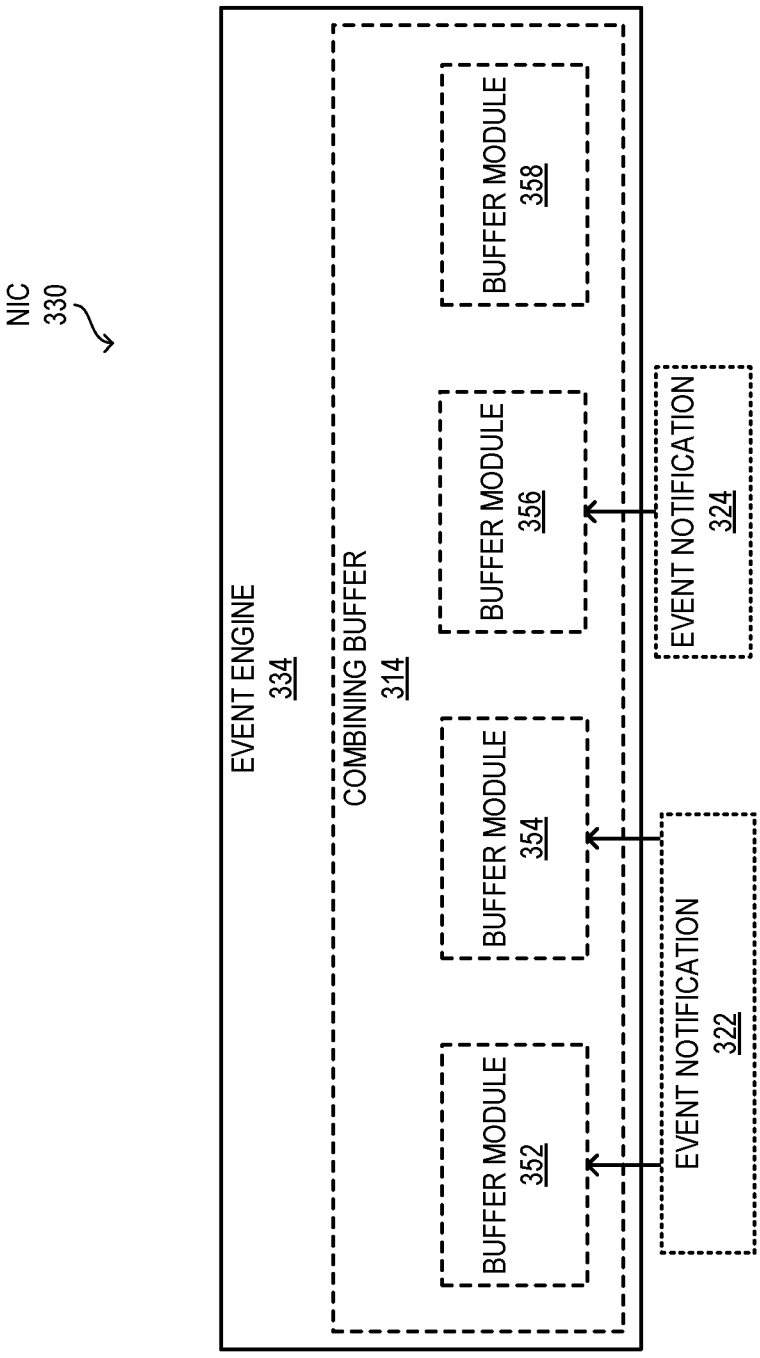
NIC
330
EVENT ENGINE
334
COMBINING BUFFER
314
BUFFER MODULE
352
BUFFER MODULE
354
BUFFER MODULE
356
BUFFER MODULE
358
EVENT NOTIFICATION
322
EVENT NOTIFICATION
324

FIG. 3B

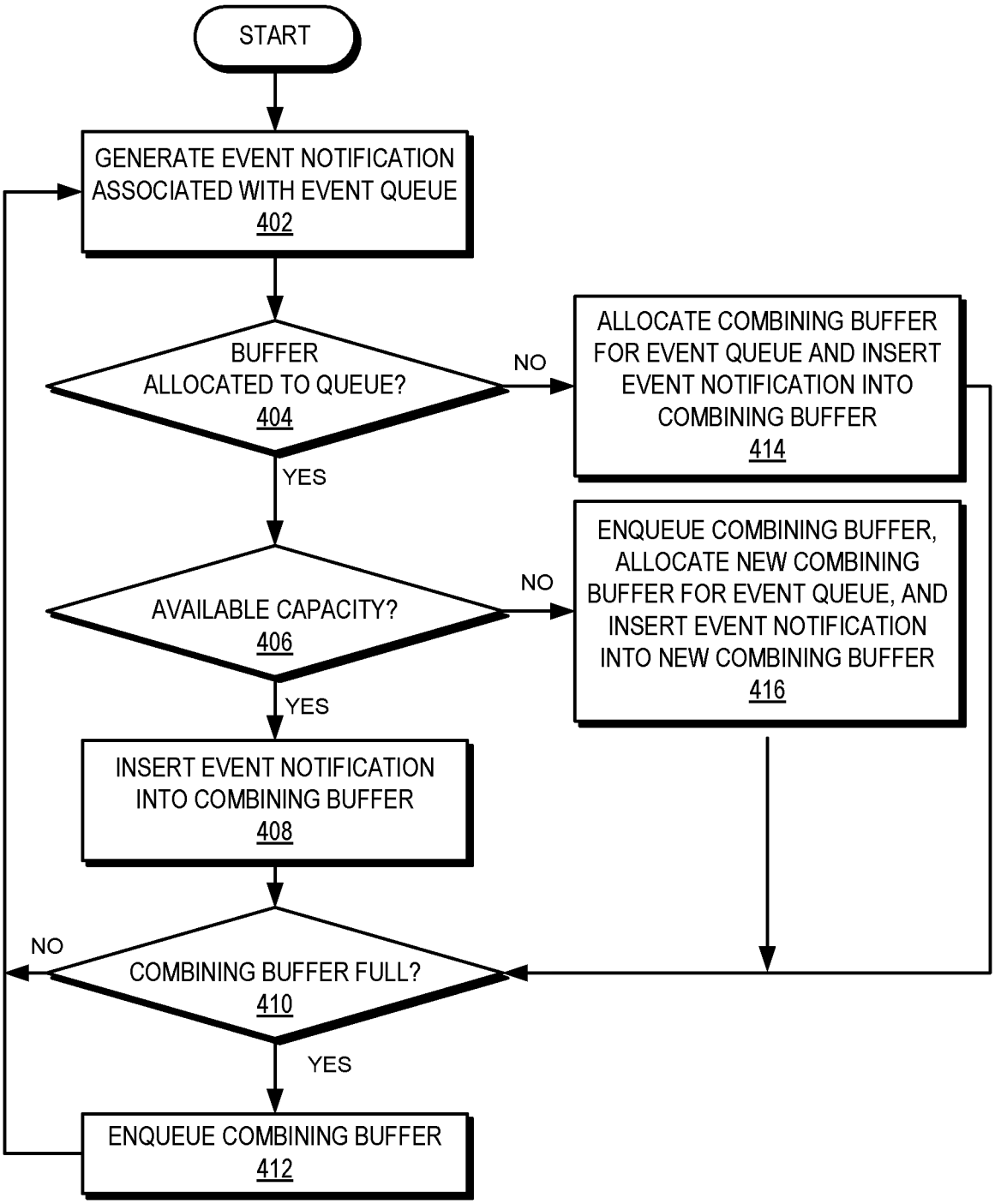
START
GENERATE EVENT NOTIFICATION ASSOCIATED WITH EVENT QUEUE
402
BUFFER ALLOCATED TO QUEUE?
404
NO
ALLOCATE COMBINING BUFFER FOR EVENT QUEUE AND INSERT EVENT NOTIFICATION INTO COMBINING BUFFER
414
YES
AVAILABLE CAPACITY?
406
NO
ENQUEUE COMBINING BUFFER, ALLOCATE NEW COMBINING BUFFER FOR EVENT QUEUE, AND INSERT EVENT NOTIFICATION INTO NEW COMBINING BUFFER
416
YES
INSERT EVENT NOTIFICATION INTO COMBINING BUFFER
408
NO
COMBINING BUFFER FULL?
410
YES
ENQUEUE COMBINING BUFFER
412

FIG. 4A

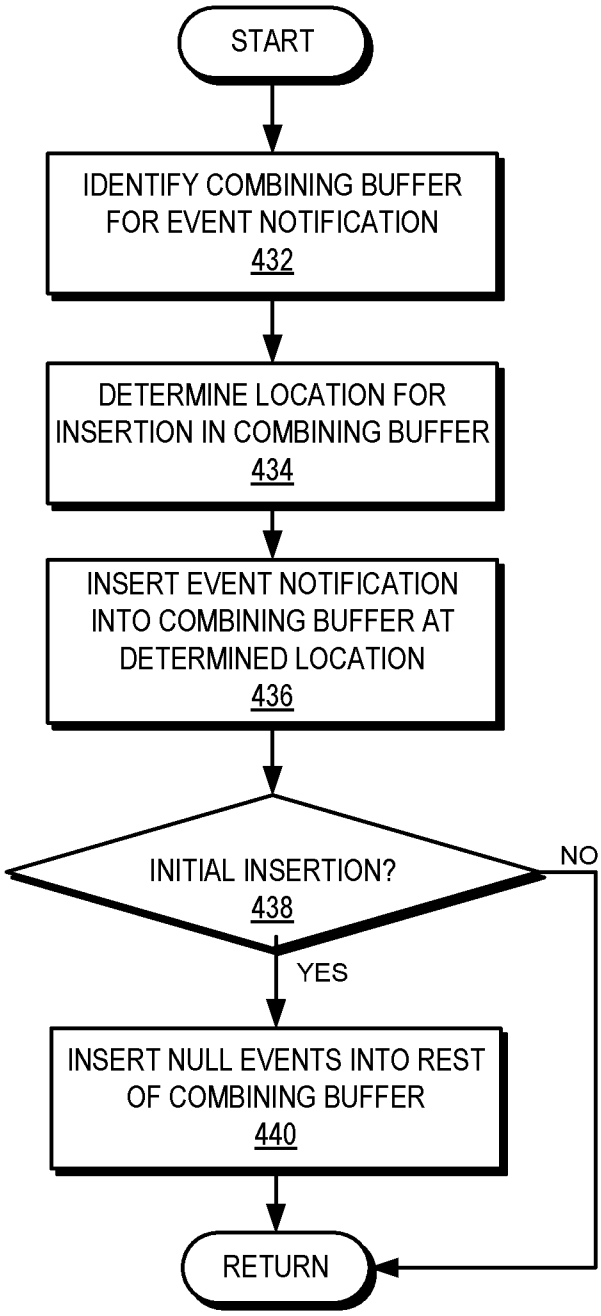
START
IDENTIFY COMBINING BUFFER FOR EVENT NOTIFICATION
432
DETERMINE LOCATION FOR INSERTION IN COMBINING BUFFER
434
INSERT EVENT NOTIFICATION INTO COMBINING BUFFER AT DETERMINED LOCATION
436
INITIAL INSERTION?
438
NO
YES
INSERT NULL EVENTS INTO REST OF COMBINING BUFFER
440
RETURN

FIG. 4B

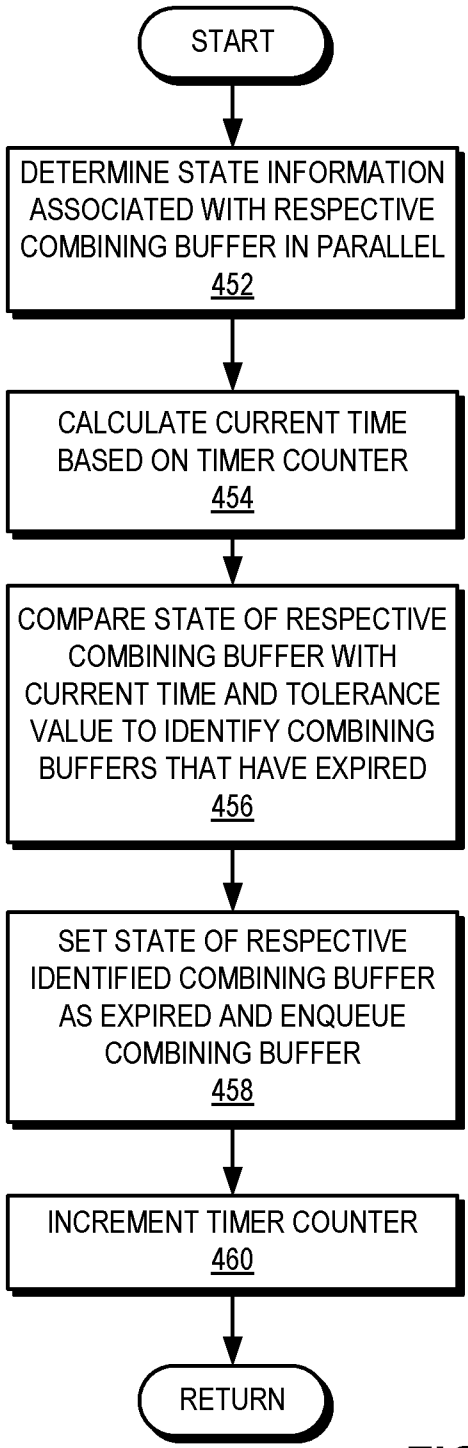
START
DETERMINE STATE INFORMATION ASSOCIATED WITH RESPECTIVE COMBINING BUFFER IN PARALLEL
452
CALCULATE CURRENT TIME BASED ON TIMER COUNTER
454
COMPARE STATE OF RESPECTIVE COMBINING BUFFER WITH CURRENT TIME AND TOLERANCE VALUE TO IDENTIFY COMBINING BUFFERS THAT HAVE EXPIRED
456
SET STATE OF RESPECTIVE IDENTIFIED COMBINING BUFFER AS EXPIRED AND ENQUEUE COMBINING BUFFER
458
INCREMENT TIMER COUNTER
460
RETURN

FIG. 4C

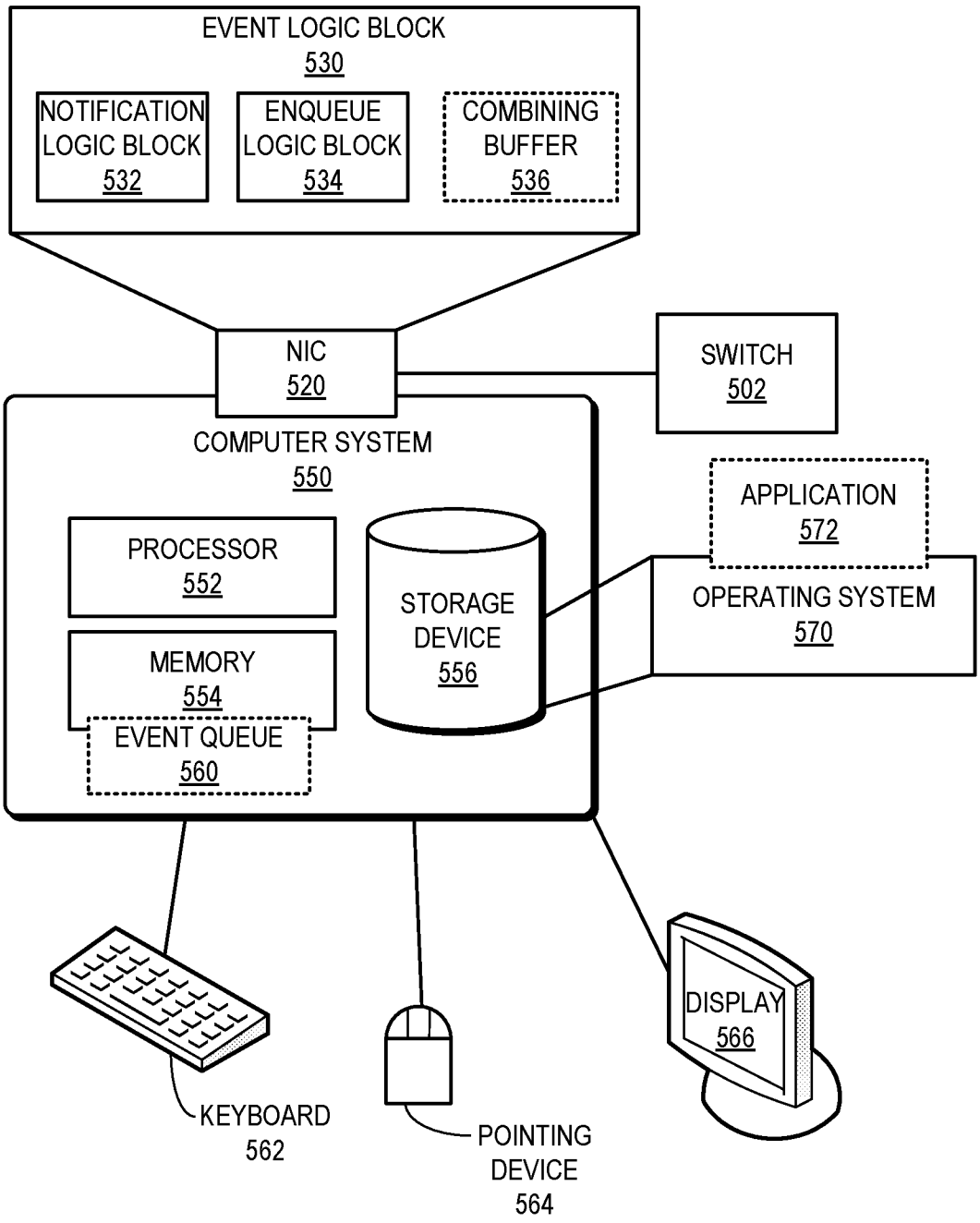
EVENT LOGIC BLOCK
530
NOTIFICATION LOGIC BLOCK
532
ENQUEUE LOGIC BLOCK
534
COMBINING BUFFER
536
NIC
520
SWITCH
502
COMPUTER SYSTEM
550
APPLICATION
572
PROCESSOR
552
STORAGE DEVICE
556
OPERATING SYSTEM
570
MEMORY
554
EVENT QUEUE
560
DISPLAY
566
KEYBOARD
562
POINTING DEVICE
564

FIG. 5

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2020/024248

A. CLASSIFICATION OF SUBJECT MATTER

H04L 12/875(2013.01)i, H04L 12/863(2013.01)i, H04L 12/861(2013.01)i, H04L 12/24(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)
H04L 12/875; G06F 21/55; G06F 21/56; G06F 9/50; H04L 12/24; H04L 12/861; H04L 12/927; H04L 29/06; H04L 12/863

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: NIC (Network Interface Controller), queue, event notification, criterion, buffer, host interface

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2019-0044809 A1 (INTEL CORPORATION) 07 February 2019 paragraphs [0003]-[0149]; claims 1, 20-21; and figures 13-19	1-20
A	US 2016-0134559 A1 (INTERNATIONAL BUSINESS MACHINES CORPORATION) 12 May 2016 paragraphs [0017]-[0037]; and figure 2	1-20
A	US 10061613 B1 (AMAZON TECHNOLOGIES, INC.) 28 August 2018 column 8, line 40 - column 22, line 12; and figure 1	1-20
A	KR 10-1850749 B1 (FUTURE INNOVATION SYSTEMS CORPORATION) 20 April 2018 claim 1; and figures 1-2	1-20
A	US 2019-0108332 A1 (ELWHA LLC) 11 April 2019 paragraphs [0119]-[0122]; and figure 3C	1-20

☐ 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
"D" document cited by the applicant in the international application
"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
07 July 2020 (07.07.2020)	**08 July 2020 (08.07.2020)**

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

Authorized officer
YANG JEONG ROK
Telephone No. +82-42-481-5709

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2020/024248

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2019-0044809 A1	07/02/2019	CN 109426629 A	05/03/2019
		CN 109428843 A	05/03/2019
		CN 109428889 A	05/03/2019
		DE 102018214774 A1	24/01/2019
		DE 102018214775 A1	06/12/2018
		DE 102018214776 A1	06/12/2018
		EP 3488670 A1	29/05/2019
		EP 3488673 A1	29/05/2019
		EP 3488674 A1	29/05/2019
		EP 3531633 A1	28/08/2019
		EP 3542270 A1	25/09/2019
		JP 2019-047489 A	22/03/2019
		TW 201804282 A	01/02/2018
		TW 201810065 A	16/03/2018
		US 10033404 B2	24/07/2018
		US 10116327 B2	30/10/2018
		US 2018-0152383 A1	31/05/2018
		US 2018-0152540 A1	31/05/2018
		US 2019-0034102 A1	31/01/2019
		US 2019-0034383 A1	31/01/2019
		US 2019-0034490 A1	31/01/2019
		US 2019-0035483 A1	31/01/2019
		US 2019-0044859 A1	07/02/2019
		US 2019-0052457 A1	14/02/2019
		US 9859918 B1	02/01/2018
		US 9954552 B2	24/04/2018
		US 9973207 B2	15/05/2018
US 2016-0134559 A1	12/05/2016	GB 2532052 A	11/05/2016
		US 10033633 B2	24/07/2018
		US 2016-0134529 A1	12/05/2016
		US 9893990 B2	13/02/2018
US 10061613 B1	28/08/2018	US 10528390 B2	07/01/2020
		US 2019-0205171 A1	04/07/2019
KR 10-1850749 B1	20/04/2018	None	
US 2019-0108332 A1	11/04/2019	None	