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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
6 March 2014 (06.03.2014)

WIPO | PCT

(10) International Publication Number
WO 2014/035377 A1

(51) International Patent Classification:
G06F 13/14 (2006.01) *G11C 16/06* (2006.01)

(21) International Application Number:
PCT/US2012/052684

(22) International Filing Date:
28 August 2012 (28.08.2012)

(25) Filing Language: English

(26) Publication Language: English

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

(72) **Inventors; and**

(75) **Inventors/Applicants** *(for US only)*: **LI, Sheng** [CN/US]; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). **YOON, Doe Hyun** [KR/US]; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US). **JOUPPI, Norman Paul** [US/US]; 1501 Page Mill Rd., Palo Alto, California 94304-1100 (US).

(74) **Agents**: **CHANG, Marcia Ramos** et al.; Hewlett-Packard Company, Intellectual Property Administration, 3404 E. Harmony Road, Mail Stop 35, 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, IS, JP, KE, KG, KM, 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, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, 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, 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))*

(54) Title: HIGH PERFORMANCE PERSISTENT MEMORY

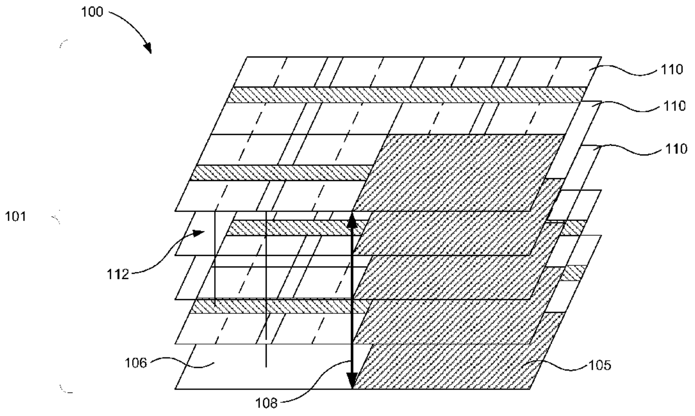

Fig. 1C

(57) **Abstract**: A method of performing data transactions in a high performance persistent memory comprising, with a processor, updating data by writing new data to non-volatile memory (NVM) and receiving a done signal from a transaction accelerator communicatively coupled to the NVM. An apparatus for high performance persistent memory, comprising a processor, a memory controller communicatively coupled to the processor, and non-volatile memory communicatively coupled to the memory controller and processor, the non-volatile memory comprising an ACID transaction accelerator, in which the processor updates data on the non-volatile memory (NVM) by writing new data to the NVM, and receives a done signal from the an ACID transaction accelerator when the data has been updated.

WO 2014/035377 A1

HIGH PERFORMANCE PERSISTENT MEMORY

BACKGROUND

[0001] Large data centers use large and relatively complex data structures. These data centers may manipulate large amounts of memory in order to process, send and receive information. One concern for modern data centers is business continuity in which a company or several companies rely on the system to run their operations. If the power provided to a data center system fails or the system crashes the company's operations may be partially impaired or operations may completely cease.

[0002] These power failures or system crashes may cause the system or an application to reboot. During a system reboot, the data center re-loads relatively complex data structures back onto the system. Data centers may load terabytes of information onto the system in order for the system to resume proper operation. Further, a system may address large amounts of data when initially loading a program. Loading such information onto the system could take several minutes or longer which may impact or stop business continuity all together.

BRIEF DESCRIPTION OF THE DRAWINGS

[0003] The accompanying drawings illustrate various examples of the principles described herein and are a part of the specification. The examples do not limit the scope of the claims.

[0004] Figs. 1A and 1B are block diagrams from a side and top view, respectively, of a memory system comprising a number of three-dimensional non-volatile memory (3D NVM) stacks according to one example of principles described herein.

[0005] Fig. 1C is a three-dimensional block diagram showing one of the three-dimensional non-volatile memory (3D NVM) stacks of Figs. 1A and 1B according to one example of the principles described herein.

[0006] Fig. 2 is a flowchart showing a method of utilizing undo and redo logging with an atomic, consistent, isolated, durable (ACID) accelerator according to one example of principles described herein.

[0007] Fig. 3 is a flowchart showing a method for undo logging with the ACID accelerator according to one example of principles described herein.

[0008] Fig. 4 is a flowchart showing a method of redo logging with the ACID accelerator according to one example of principles described herein.

[0009] Fig. 5A and 5B are accelerator designs for undo logging and redo logging, respectively, according to one example of principles described herein.

[0010] Fig. 6 is a flowchart showing a method of scheduling memory between a memory controller and an ACID accelerator and efficiently writing data to NVM according to one example of the principles described herein.

[0011] Throughout the drawings, identical reference numbers designate similar, but not necessarily identical, elements.

DETAILED DESCRIPTION

[0012] The present specification describes a method of performing data transactions in high performance persistent memory comprising, with a processor, updating data by writing new data to non-volatile memory (NVM) and receiving a done signal from a transaction accelerator communicatively coupled to the NVM.

[0013] The present specification further describes an apparatus for high performance persistent memory, comprising a processor, a memory controller communicatively coupled to the processor, and non-volatile memory communicatively coupled to the memory controller and processor, the non-volatile memory comprising an ACID transaction accelerator, in which the processor updates data on the non-volatile memory (NVM) by writing new data to the NVM, and receives a done signal from the ACID transaction accelerator when the data has been updated.

[0014] The present specification also describes a computer program product for performing ACID transactions in a high performance persistent memory device. The computer program product may comprise a computer readable storage medium comprising computer usable program code embodied therewith. The computer usable program code may comprise computer usable program code to, when executed by a processor, update data by writing new data to non-volatile memory (NVM) and receive a done signal from a transaction accelerator communicatively coupled to the NVM.

[0015] As noted above, large data centers use large and relatively complex data structures. These data centers may manipulate a large amount of memory in order to process, send and receive information. One concern for modern data centers is business continuity in which a company or several companies rely on the system to run their operations. If the power provided to a data center system fails or the system crashes, the company's operations may be partially impaired or operations may completely cease. Consequently, these power failures or system crashes may cause the system or a program running on the system to reboot. During a system reboot, the data center re-loads relatively complex data structures back onto the system. Data centers may load terabytes of information onto the system in order for the system to resume proper operation. Further a system may address large amounts of data when initially loading a program. Loading such information onto the system could take several minutes or longer which may impact or stop business continuity all together.

[0016] In order to load these large and relatively complex data structures, a high performance persistent memory system may be used to process that large amount of data in a quick, inexpensive, and efficient manner. Accomplishing this, the large and complex data structures may be ready for use when a program starts or after the program or system reboots.

[0017] In one example of the present description, the 3D NVM achieves a much higher performance than existing implementations. This is accomplished by maintaining checkpointing locally in the NVM without the complex undo and redo log constraints. Thus, if a system using high performance persistent memory, as described herein, loses power, the program hangs, or the system crashes, the last transaction is used as a checkpoint to restore system data. In one example, the 3D NVM may provide hardware support for separating cache systems from durability to achieve inexpensive universal persistent memory without forfeiting performance and programming flexibility with minimal changes to the processor and operating system's architecture.

[0018] In various examples, a high performance persistent memory system described herein is used for data centers with relatively large in-memory data sets. Often large amounts of memory are loaded onto a computer system. This data may be used to, for example, load a large operating system comprising of a complex data structure onto a computer when the computer is initially powered on. Additionally, this data may include relatively complex data structures that provide functionality for a program. The present high-performance persistent memory system leverages a number of 3D NVMs with a logic stack to quickly access data after a crash without reading bytes serially from memory and building data structures in the memory.

[0019] As used in the present specification and in the appended claims, the term "high performance persistent memory“ is meant to be understood broadly as fast access non-volatile memory (NVM) that can retain and store information even when the power to the device is no longer available. High performance persistent memory may therefore retain data if and when a program running on the system is disrupted or the system experiences a drop in power.

[0020] Additionally, as used in the present specification and in the appended claims the term "three-dimensional non-volatile memory (3D NVM)" refers broadly to any memory storage medium wherein data can be stored and retrieved. In one example, the 3D NVM may not require power to sustain the information stored thereon. Still further, in one example, a number of 3D NVMs may be stacked on top of each other allowing for vertical expansion of the high performance persistent memory.

[0021] Further, as used in the present specification and in the appended claims, the term “logic die“ is meant to be understood broadly as a small block of semiconducting material on which functional integrated circuits are fabricated. In one example, the logic die provides architecture support for the high persistent memory.

[0022] Still further, as used in the present specification and in the appended claims, the term “logical operation” is meant to be understood as any operation involving the use of logical functions, such as “AND” or “OR”, that are applied to the input signals of a particular logic circuit. A logical operation may also be referred to as a “transaction.”

[0023] Even further, as used in the present specification and in the appended claims, the term “ACID transaction“ is meant to be understood broadly as any set of transaction properties that provide that a transaction sent to the database is processed reliably. In one example, a set of properties are defined for each transaction such that they are atomic, consistent, isolated and durable (ACID).

[0024] For a transaction to be “atomic” each transaction is entirely commited. Therefore, if a transaction is “atomic” then, when one part of the transaction fails, the whole transaction will fail and the state of the non-volatile memory will remain unchanged.

[0025] Further, for a transaction to be “consistent” each transaction made will bring the database from one valid state into another valid state. Any data written to a database is assured to be valid for all predefined rules. These rules may include, but are not limited to cascades, triggers, or constraints. For example, if a transaction is requested and the system process determines the transaction will move data into an invalid state the transaction is not executed.

[0026] Additionally, for a transaction to be “isolated” this property ensures that if a number of transactions were to be executed instead of sequentially, the result will comprise the same system state as if the transactions were executed serially. Thus, any one transaction executed before, after, or concurrently with another transaction, each will result in the same state.

[0027] Further, for a transaction to be “durable,” once a transaction is committed, it will remain committed or stored permanently, even in the event of power loss or system crash. Thus, the transaction is non-volatile and persistent.

[0028] Examples of the present system and method are described below with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems) and computer program products. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer program code. This computer program code may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the code, which executes via a processor of the computer or other programmable data processing apparatus, to implement the functions/acts specified in the flowchart(s) and/or block diagram block or blocks.

[0029] In one example, this computer program code may be stored in a computer-readable storage medium that can direct a computer or other programmable data processing apparatus to function in a particular manner, such that the code stored in the computer-readable memory produces an article of manufacture including program code which implements the functions/act specified in the flowchart(s) and/or block diagram blocks or blocks.

[0030] The computer program code may also be loaded onto a computer or other programmable data processing apparatus to cause a series of operations to be performed on the computer or other programmable apparatus to produce a computer implemented process such that the computer code which executes on the computer or other programmable apparatus implements the functions/acts specified in the flowchart(s) and/or block diagram blocks or blocks.

[0031] In the following description, for purposes of explanation, numerous specific details are set forth in order to provide a thorough understanding of the present systems and methods. It will be apparent, however, to one skilled in the art that the present apparatus, systems, and methods may be practiced without these specific details. Reference in the specification to "an example" or similar language means that a particular feature, structure, or characteristic described in connection with that example is included as described, but may not be included in other examples.

[0032] Referring now to the Figures, Fig. 1A shows a side view block diagram of a memory system (100) comprising a number of three-dimensional non-volatile memory (3D NVM) stacks (101) according to one example of principles described herein. As illustrated in the system (100), the 3D NVM stacks (101) may include a number of vertically placed slices of non-volatile memory (NVM) (110) comprising multiple NVM dies. Other examples of memory which may be used may include memory devices such as ROM, nvSRAM, FeRAM, MRAM, PRAM, CBRAM, SONOS, NRAM or other types of non-volatile memory. Therefore, although Fig, 1 shows a number of vertically stacked NVRAM (110) devices, the NVM devices may incorporate any type of non-volatile memory, NVRAM being an example.

[0033] Additionally, instead of the NVM being in the form of 3D NVM stacks (101), the NVM memory may instead be positioned in a two-dimensional configuration. Therefore, although Figs. 1A, 1B, and 1C show the NVM stack (101) being three-dimensional, any memory configuration may be used in the present description without diverging from the principles described herein.

[0034] The vertically placed NVRAM devices (110) may be stacked on each other to produce a 3D stack (101) of NVMRAM devices (110). Each NVRAM device (110) within each of the 3D NVM stacks (101) may be communicatively coupled to a number of other NVRAM devices (110) in the 3D NVM stack (101) via a through-silicon via (TSV) (112, Fig. 1C) created in each of the NVRAM devices (110) during the manufacturing process. The TSVs (112, Fig. 1C) may act as a bus to allow all of the NVRAM devices (110) within the 3D NVM stacks (101) to behave as a single device.

[0035] In one example, the 3D NVMRAM stacks (101) may be used to build simple memory modules or to build scalable memory networks. Although Fig. 1 shows a number of vertically placed slices of NVRAM (110) stacked together forming a 3D NVRAM (101), the present specification contemplates that any number and type of NVM may be communicatively coupled together either horizontally or vertically. Stacking of the number of NVRAM devices (110) of may have a number of advantages. One advantage is that physical space within a computing system (100) is saved by taking advantage of the vertical space available above the memory board. The system (100) may therefore involve as few or as many NVRAM devices (110) in order for the system to operate.

[0036] The 3D NVRAM stacks (101) may receive data from a processor (102) and be directed to store the data thereon. Additionally, a memory controller (Fig. 1B, 103) may be used to manage the flow of data moving to and from each of the NVRAM devices (110) in the 3D NVM stacks (101).

[0037] Fig. 1B shows a top view block diagram of a memory system (100) comprising a number of three-dimensional non-volatile memory (3D NVM) stacks (101) according to one example of principles described herein. As discussed above, the NVRAM devices (110) may be controlled by a memory controller (103) that manages the data flow in and out of the 3D NVM stacks (101). Communication between the NVRAM devices (110) and the memory controller (103) may be accomplished by using routing interconnects (111) on a silicon interposer (104). In one example, the individual NVRAM devices (110) or three-dimensional non-volatile memory (3D NVM) stacks (101) may not be included on the same silicon interposer (104) and instead may be physically distant form the processor (102) and memory controller (103) while still being communicatively coupled to them via an interconnect (111).

[0038] In operation, the processor (102) may send executable code to the memory controller (103) so that the memory controller can manage the data flow to the individual NVRAM devices (110). In one example, the processor (102) may send updated data to the number of NVMRAM devices (110).

[0039] Fig. 1C is a three-dimensional block diagram showing one of the number of three-dimensional non-volatile memory (3D NVM) stacks (101) of Figs. 1A and 1B according to one example of the principles described herein. Each vertically placed NVRAM device (110) may comprise portions of multiple NVM dies and may form single rank or multiple rank channels (108) between each NVRAM device (110). An ACID transaction accelerator (105) may be communicatively coupled to each of NVRAM devices (110) as well as on the logic die (106). In one example, the ACID transaction accelerator (105) may be physically coupled to the NVM such that it is placed on the logic die onto which the NVM devices (110) are also coupled. In another example, the ACID transaction accelerator (105) can physically exist apart form the logic die (106). Therefore, although Fig. 1C may show that the ACID transaction accelerator (105) is placed on a three-dimensional stack of NVM devices, other examples exist where the ACID transaction accelerator (105) is communicatively coupled to the NVM devices, but placed on its own logic die.

[0040] The transaction accelerator (105) is used to maintain atomic, consistent, isolated, and durable transactions as described above. Additionally, the accelerator (105) may ensure that minimal changes are made to the processor and operating system architecture of the system (100).

[0041] Fig. 2 is a flowchart showing a method of utilizing undo and redo logging using an ACID accelerator (105) according to one example of principles described herein. The method may begin by issuing an update (201) command, for example, by an operator, system, or device. Here the new data may be written to the NVM (201) according to the ACID properties mentioned above. During this process (200), the accelerator (105) may use a checkpointing technique to, with the current data in the NVM (110), store the current state of data being transferred. If, according to any of the ACID transaction properties, the update process or the transaction process fails and the new data is not written to the NVM, this checkpointing procedure will allow the system (100) to be able to restart at the point of failure.

[0042] As will be described below, the accelerator (105) may be given access to a number of buffers which contain new data received from the processor (102) and old data retained by the NVRAM device (110). Control logic may be used by the accelerator (105) to read the old data, log data to the NVRAM device (110), wait until the logging finishes, and write the buffered new data to the NVRAM device (110). During this process, however, the durability property is separate from the writing data process. In one example, by buffering the data in a new data buffer and an old data buffer on the accelerator (105), the memory operations may be optimized through bulk data processing.

[0043] In this case, updating the data to the NVRAM devices (110) by having the processor (102) read the old data, pushing a tuple comprising the address of the old data to an undo log, waiting until the tuple is written out to the undo log, and writing the new data to the 3D NVM stacks (101) need not happen. Here, it can be appreciated that the NVM will be accessed for both the log operation and the data updates.

[0044] Instead, the memory controller (103) as described in the present specification may simply write the new data to the 3D NVM stacks (101) and wait until all the data in the transaction is written out to the 3D NVM stacks (101). The ACID requirement that the transaction be durable is separated from the data access process and the system may provide a high performing, yet fast and cheap persistent memory system (100). Additionally, the logging operation is transparent to the processor (102) and the processor (102) will treat the transaction updates as regular memory updates.

[0045] Once all of the data has been written to the NVM a done signal will be received (202) from the accelerator (105). Thus, the ACID transaction is now stored in the appropriate 3D NVM stack (101). If a system (100) were to fail or lose power, the accelerator (105) can recognize the most recent version of persistent data.

[0046] Fig. 3 is a flowchart showing a method (300) for undo logging with the ACID accelerator (105) according to one example of principles described herein. Fig 3 shows how the system (100) of Figs. 1A, 1B, and 1C completes ACID transactions as an undo logging transaction. The ACID transaction begins when the accelerator receives (301) new data from the processor (102). The old data is then read (302). The ACID accelerator then logs (303) bulk data the NVM. The bulk data may be defined as buffered old data with addresses defining where within the NVRAM devices (110) the data was stored. Using the bulk data that is buffered helps to optimize memory operations where write and wait time is optimized in the stacked NVM since there is no roundtrip delay between the NVM and memory controller. The system then waits (304) until logging is finished. Once logging has finished the buffered new data is written (305) the NVM.

[0047] The buffers within the accelerator (105) can be memory managed by the controller (103) or can be a cache like structure with hardware managed tag and metadata in addition to data blocks. Additionally, the accelerator (105) may perform multiple loggings for a transaction, or may handle multiple transactions at the same time.

[0048] As noted above, data may be reordered to improve the channel utilization, and the ACID accelerator (105), by buffering incoming data, may reconstruct the correct ordering. The processor (102) may direct the memory controller (103) to send metadata defining the order of the data along with the data and transaction ID. This metadata may be sent to the accelerator (105) via an express bus created between the last level cache and the controller (103). This bus may be dedicated to sending a write-reservation that includes the time stamp and transaction ID. Since the data to be sent over this bus includes meta-data, it may be relatively smaller than data of real memory accesses. Thus, the extra bus will incur minimal pressure on processor pin count. When the data write is complete a done signal is received from the accelerator (306) by the processor (102). Advantageously, any new data is written out to NVM after the old data is pushed to the undo log. Thus, serialization may be avoided in the architecture of the present example during undo logging.

[0049] Therefore, the present system (100) may allow memory writes of transactions to be issued from memory controller (103) out-of-order as if they were normal memory writes so as to maintain a high performance level. While the buffers within the ACID accelerator (105) can buffer and reorder the memory writes with the metatdata to maintain the correct order with regards to transactions it is also possible for the buffers to be filled up with partially updated transactions. In other systems this may prevent a number of transactions from moving forward and the systems may be dead-locked. However, the present ACID accelerator (105) may place a threshold limit on how many partially committed transactions and their data can be queued up in the buffers. This threshold limit may be defined by the system (100) to fit any particular set of transactions or may be user defined.

[0050] Since the accelerator (105) is aware of how many transactions have been issued and how many cache lines have been updated based on the metadata stored in the accelerator's buffers and provided by the processor-side memory controller (103), the accelerator (105) may request the memory controller (103) to flush the dirty cache lines of the finished transactions (i.e. transactions not committed to the NVM (110)). In this case, the memory controller (103) may not be allowed to issue the memory required at will and based on its own scheduling policy. Through careful co-operation between the memory controller (103) and the ACID accelerator (105), the system (100) may be able to support persistent memory with minimal performance penalty and avoid any potential dead-locks. In one example, this persistency-aware memory scheduling may be implemented based on whether the instant durability is needed. In one example, the memory controller (103), processor (102), and operating system can also choose whether to allow memory writes to be issued out-of-order or just flush the data to the NVM (101) as soon as it may be allowed.

[0051] An example of the undo logging process according to the present application will now be described. Assume 5 transactions are sent to the ACID accelerator (105), namely; A, B, C, D, and E. Originally, these transactions are committed in alphabetical order. Further assume that these transactions produce the following data blocks, namely; A1, A2, A3, B4, B5, B6, C7, C8, D9, and E10. As described above, the transaction accelerator (105) may receive a number of write-reservations with time stamps and transaction IDs. In this way, the transaction accelerator (105) is notified of the fact that, for example, transaction A has 3 memory write blocks, A1, A2, and A3. Further assuming, the processor or memory controller (103) reorders the data writes so the NVM receives the following sequence: A1, E10, A2, B4, B5, C7, A3, C8, D9, B6. As described above the incoming data is first buffered (303). When A3 is received, the accelerator commits A1, A2, and A3 to NVM and a done signal is received from the accelerator (306). Further, B is not committed until B6 is received so the transactions B, C, D, and E are buffered. When B6 is received, the last transaction in the set, the ACID accelerator (105) has all the data for transactions B, C, D, and E. Consequently, serialization is avoided and all the transactions are then committed at the same time.

[0052] Fig. 4 is a flowchart showing a method for redo logging with the ACID accelerator (105) according to one example of principles described herein. Fig 4 shows how the system (100) of Figs. 1A, 1B, and 1C completes ACID transactions as a redo logging transaction. The ACID transaction begins wherein the accelerator (105) receives (401) new buffered data from the processor (102). No further action is performed immediately until the last data write for the transaction is sent (402) to the accelerator (105). After all the new buffered data has been received by the accelerator (105), the bulk data is logged (403) to the NVM. Once logging (403) has been finished (404) a done signal is received (405) from the accelerator (105). When the done signal is received (405), the new buffered data is written (406) to the NVM.

[0053] Similar to above, the accelerator (105) may perform multiple loggings for a transaction, or may handle multiple transactions at the same time. Additionally, new data may be written (406) out to the NVM after the transaction finishes and the whole redo logging for the transaction is finished (404). Also, similar to undo logging, the ACID accelerator (105) can provide a relatively simpler interface by optimizing the memory operation with the bulk data processing by buffering the data, writing it, and waiting. This proves to be a much faster process within the stacked memory since there is no roundtrip delay between the 3D NVM stacks (101) and the memory controller (103). Still further, the processor can have the same interface, while the NVM stack chooses the optimal approach; namely undo logging (300) or redo logging (400) for ACID support.

[0054] Fig. 5A is an illustration of an accelerator (500) design for undo logging according to one example of principles described herein. Fig 5A shows a 3D NVM stack (101) within the system (100) of Figs. 1A, 1B, and 1C with an accelerator (500) design for undo logging transaction. Undo logging provides for a logic controller (501) which may include hardware logic and a processor executing computer usable program code. Here, the controller (501) may produce the desired logic for the system. As noted above, both the new data and the old data is to be buffered when undo logging is desired and is written to NVM (504). Fig. 5A shows that the new data and old data may be stored, at least temporarily in a new (502) and old data buffer (503) respectively. These buffers (502, 503) may be reused once a consistent and/or persistent version of the data being updated has been created in the NVM (504). In one example, the operating system associated with the computing system and NVM (504) may help to allocate portions of the NVM (504). In some examples, different portions of the NVM (504) may be allocated to fit a variety of different transactions that may take place in connection with the NVM (504).

[0055] As discussed above, the system (100) may allow memory writes to be issued from memory controller (103) to the NVM (504) out-of-order as if they were normal memory writes. While the number of buffers (502, 503) within the ACID accelerator (105, 500) can buffer and reorder the memory writes with the metadata provided from the memory controller (103), it is possible for the number of buffers (502, 503) to be filled up with partially updated transactions. The ACID accelerator (105, 500), however, may place a threshold limit on how many partially committed transactions and their data can be queued up in the buffers. This threshold limit may be defined by the system (100) to fit any particular set of transactions or may be user defined.

[0056] Since the accelerator (105, 500) is aware of how many transactions have been issued and how many cache lines have been updated based on the metadata provided by the processor-side memory controller (103), the accelerator (105, 500) may request the memory controller (103) to flush the dirty cache lines of the finished transactions. In this case, the memory controller (103) may not be allowed to issue the memory required at will based on its own scheduling policy. Through co-operation between the memory controller (103) and the ACID accelerator (105, 500), the system (100) may be able to support persistent memory with minimal performance penalty. In one example, this persistency-aware memory scheduling may be implemented based on whether the instant durability is needed. In one example, the memory controller (103), processor (102), and operating system can also choose whether to allow memory writes to be issued out-of-order or just flush the data to the NVM (110) as soon as it may be allowed.

[0057] In one example, the ACID accelerator (105, 500), using the control logic (501) within the ACID accelerator (105, 500), may control the interfacing between the number of buffers (502, 503) and the NVM (110). In one example, as the number of buffers (502, 503) begin to fill up, the ACID accelerator (105, 500) will complete the log transactions in order to make sure that data is persistently logged when appropriate and as soon as possible. However, once any log transaction is completed, the ACID accelerator (105, 500) may write bulk data to the NVM (110) when appropriate. For example, if the target memory block within the NVM (110) may be busy when the ACID accelerator (105, 500) is attempting to write to that memory block, the ACID accelerator (105, 500) may first commit other transactions to the NVM (110) until that memory block becomes available. In this way, the ACID accelerator (105, 500) may take advantage of time that would have otherwise been spent waiting for busy memory blocks to complete other transactions.

[0058] Fig. 5B is an illustration of an accelerator (105) design example for undo logging according to one example of principles described herein. Fig 5B shows a 3D NVM stack (101) within the system (100) of Figs. 1A, 1B, and 1C with an accelerator (105) design for redo logging transaction. Redo logging provides for a logic controller (501) which may be hardware logic or a simple processor with computer usable program code embodied thereon. In either case the controller (501) is able to produce the desired logic for the system (100). As noted above, the new data (502) is to be buffered when redo logging is initiated and is written to the NVM (504).

[0059] Fig. 6 is a flowchart showing a method (600) of scheduling memory between a memory controller (103) and an ACID accelerator (105, 500) as well as a method for efficiently writing data to the NVM (101) according to one example of the principles described herein. Although the two methods depicted here in Fig. 6 (i.e. the method of scheduling memory between a memory controller (103) and an ACID accelerator (105, 500) and the method for efficiently writing data to NVM (101)) are shown together as method 600, the present specification also contemplates that these two methods may be started and occur separately and independently of each other. Therefore, Fig. 6 is meant to be understood as being merely an example of the methods described herein.

[0060] While the processor and memory controller keep generating memory requests from both transactions and normal writes, the ACID accelerator (105, 500) may make a decision (610) as to whether a threshold limit on the number of partially committed transactions has been met. If the threshold has been met (Determination Yes, 610), the ACID accelerator (105, 500) may notify (650) the memory controller (103) to stop sending data of new transactions and request (655) that the memory controller (103) flush the dirty cache lines of the finished transactions. The ACID accelerator (105, 500) may then complete (660) a number of partially updated transactions by performing the logging and updating steps (605, 615, 620, 625, 630, 635, 645, 640) as mentioned below. Once this occurs, the ACID accelerator (105, 500) may then again determine (610) if the threshold limit on the number of partially committed transactions has been met. Therefore, in one example, the ACID accelerator (105, 500) may continually check after each completion of a partially updated transaction, whether the threshold limit has still been reached. In another example, the ACID accelerator (105, 500) may complete a predetermined number of partially updated transactions and then make the same query (610).

[0061] If the threshold has not been met (Determination No, 610), the ACID accelerator (105, 500) may complete the method of writing data to the NVM (110) by continuing to let the memory controller (103) issue a number of memory requests at will and accept (605) new data from the memory controller (103). As described above, the new data received (605) may be out-of-order.

[0062] The ACID accelerator (105, 500) may then read (615) the old data as described above. After reading (615) the old data, the ACID accelerator may then log (620) bulk data the NVM. A determination may then be made (625) as to if the data block that is to be written to is busy. If the data block is busy (Determination Yes, 625), then the ACID accelerator (105, 500) may commit (645) other transactions to the NVM and wait for the data block to become available. In this case, when the data block does become available, the process continues with the ACID accelerator (105, 500) waiting (630) until logging is finished, writing (635) the buffered new data to the NVM (110), and sending (640) a done signal to the processor (102) and memory controller (103).

[0063] If the data block is not busy (Determination No, 625), then the ACID accelerator (105, 500) waits (630) until logging is finished. The ACID accelerator (105, 500) then writes (635) the buffered new data to the NVM (110) and sends (640) a done signal to the processor (102) and memory controller (103). The whole process may then repeat throughout the execution of applications.

[0064] Although Fig. 6 above describes a method of scheduling memory between a memory controller (103) and an ACID accelerator (105, 500) and efficiently writing data to the NVM (110), in one example, the method may include only the method of scheduling memory between a memory controller (103) and the ACID accelerator (105). In another example, the method may include only the method of efficiently writing data to the NVM (110) as described above.

[0065] The present specification may also be described as a computer program product for performing ACID transactions in a high performance persistent memory device. The computer program product may comprise a computer readable storage medium comprising computer usable program code embodied therewith. The computer usable program code may comprise computer usable program code to, when executed by a processor, update data by writing new data to non-volatile memory (NVM) and computer usable program code to, when executed by a processor, receive a done signal from a transaction accelerator communicatively coupled to the NVM.

[0066] Any combination of computer readable medium(s) may be utilized in the present specification. A computer readable storage medium may be, for example, but not limited to, an electronic, magnetic, optical electromagnetic, infrared, or semiconductor system, apparatus, or device or any suitable combination of the foregoing. More specific examples (a non-exhaustive list) of the computer readable mediums would include the following: an electrical connection having a number of wires, a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROP or Flash memory), an optical fiber, a portable compact disk read-only memory (CD-ROM), an optical storage device, a magnetic storage device, or any suitable combination of the foregoing. In the context of this document, a computer readable storage medium may be any tangible medium that can contain, or store a program for use by or in connection with any instruction execution system, apparatus, or device such as, for example, a processor.

[0067] Program code embodied on a computer readable medium may be transmitted using any appropriate medium, including but not limited to wireless, wireline, optical fiber cable, RF, etc., or any suitable combination of the foregoing.

[0068] Computer program code for carrying out operations of the present specification may be written in an object oriented programming language such as Java, Smalltalk, or C++, among others. Computer program code for carrying out operations of the present specification may also be written in declarative programming language such as Structured Query Language, However, the computer program code for carrying out operations of the present systems and methods may also be written in procedural programming languages, such as, for example, the “C” programming language or similar programming languages. The program code may execute entirely on the user’s computer, partly on the user’s computer, as a stand-alone computer readable medium package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, thought the internet using an internet service provider).

[0069] The flowchart and block diagrams in the figures illustrate the architecture, functionality, and operations of possible implementations of systems, methods, and computer program products. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of code, which comprises a number of executable instructions for implementing the specific logical function(s). It should also be noted that, in some alternative implementations, the functions noted in the block may occur out of the order noted in the figures. For example, two blocks shown in succession may, in fact, be executed substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustrations and combination of blocks in the block diagrams and/or flowchart illustrations, can be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

[0070] The terminology used herein is for the purpose of describing particular examples, and is not intended to be limiting. As used herein, the singular forms "a," "an" and "the" are intended to include the plural forms as well, unless the context clearly indicated otherwise. It will be further understood that the terms "comprises" and/or "comprising" when used in the specification, specify the presence of stated features, integers, operations, elements, and/or components, but do not preclude the presence or addition of a number of other features, integers, operations, elements, components, and/or groups thereof.

[0071] The preceding description has been presented to illustrate and describe examples of the principles described. This description is not intended to be exhaustive or to limit these principles to any precise form disclosed. Many modifications and variations are possible in light of the above teaching.

CLAIMS

WHAT IS CLAIMED IS:

1. A method of performing data transactions in a high performance persistent memory comprising

with a processor:

updating data by writing new data to non-volatile memory (NVM); and

receiving a done signal from a transaction accelerator communicatively coupled to the NVM.

2. The method of claim 1, in which updating data comprises receiving new data from the processor at the transaction accelerator and temporarily buffering the new data in a new data buffer associated with the transaction accelerator.

3. The method of claim 2, in which the new data received comprises a number of partially committed transactions and in which, upon exceeding a threshold limit of the number of partially committed transactions in the buffer, the transaction accelerator instructs the processor to flush a number of dirty cache lines of a number of finished transactions.

4. The method of claim 3, in which, if a number of data blocks within the NVM to which the transaction accelerator is attempting to write a completed transaction to is busy, the transaction accelerator commits a number of other transactions to the NVM until the data block becomes available.

5. The method of claim 2, in which updating data comprises buffering new data received from the processor and old data retained on the non-volatile memory.

6. The method of claim 1, in which writing new data to NVM comprises writing the new data to NVM using bulk data processing.

7. The method of claim 1, in which receiving the done signal from the accelerator comprises sending a done signal from the accelerator to the processor when a data write is complete.

8. The method of claim 1, further comprising receiving metadata from a memory controller communicatively coupled to the processor defining the number and order of writes to the new data.

9. The method of claim 6, in which the metadata is received by the transaction accelerator from the memory controller via a dedicated bus communicatively coupling the memory controller to the NVM.

10. An apparatus for high performance persistent memory, comprising:

a processor;

a memory controller communicatively coupled to the processor; and

a non-volatile memory communicatively coupled to the memory controller and processor, the non-volatile memory comprising an ACID transaction accelerator;

in which the processor:

updates data on the non-volatile memory (NVM) by writing new data to the NVM; and

receives a done signal from the an ACID transaction accelerator when the data has been updated.

11. The apparatus of claim 10, in which the ACID accelerator, when instructed by the processor:

reads old data;
logs the old data to NVM; and
writes buffered new data to NVM.

12. The apparatus of claim 10, in which the memory controller is communicatively coupled to the NVM via a dedicated bus, and in which the memory controller sends to the NVM metadata defining the number and order of writes made to the new data.

13. The apparatus of claim 10, in which the ACID accelerator sends a done signal to the processor when the data has become persistent.

14. The apparatus of claim 10, in which the new data received comprises a number of partially committed transactions and in which, upon exceeding a threshold limit of the number of partially committed transactions in the buffer, the transaction accelerator instructs the memory controller to flush a number of dirty cache lines of a number of finished transactions.

15. A computer program product for performing ACID transactions in a high performance persistent memory device, the computer program product comprising:

a computer readable storage medium comprising computer usable program code embodied therewith, the computer usable program code comprising:

computer usable program code to, when executed by a processor, update data by writing new data to non-volatile memory (NVM); and

computer usable program code to, when executed by a processor, receive a done signal from a transaction accelerator communicatively coupled to the NVM.

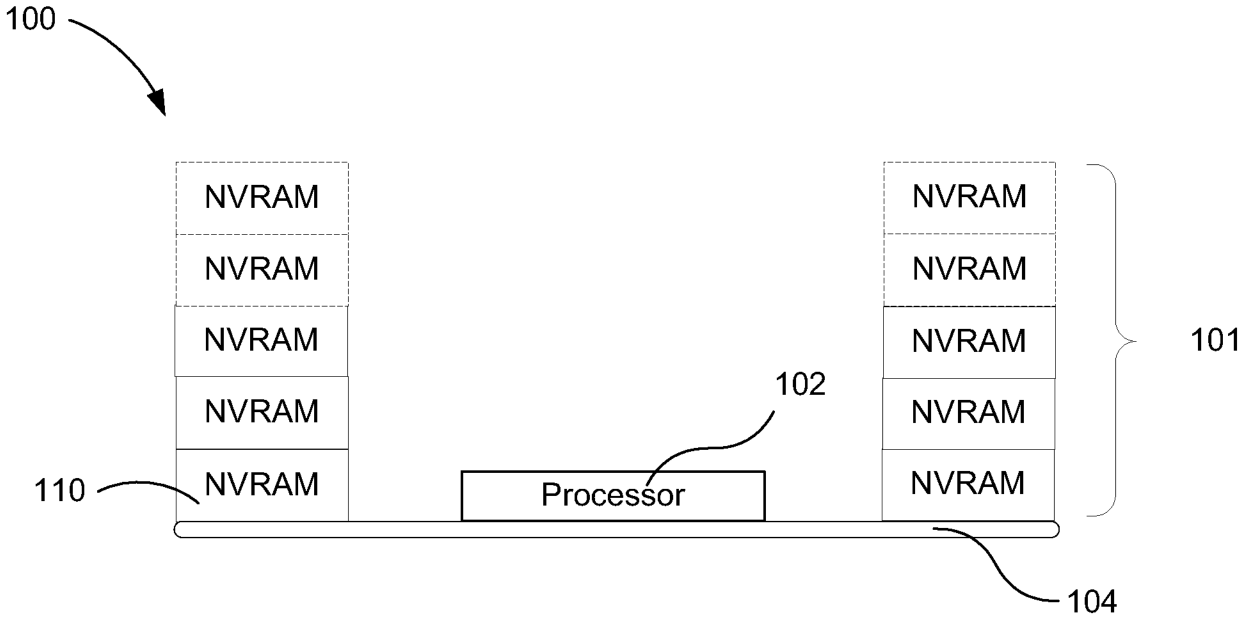
100
NVRAM
NVRAM
NVRAM
NVRAM
NVRAM
NVRAM
NVRAM
NVRAM
NVRAM
NVRAM
101
102
110
Processor
104

Fig. 1A

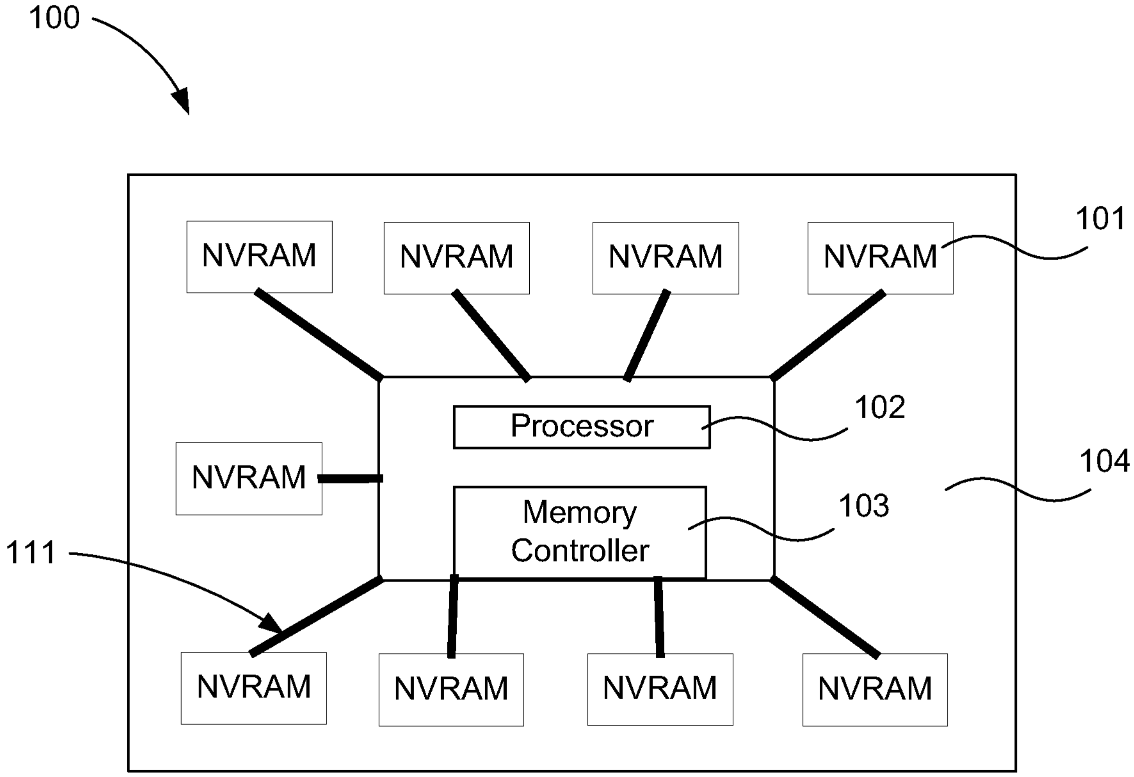
100
NVRAM
NVRAM
NVRAM
NVRAM
101
Processor
102
NVRAM
104
Memory Controller
103
111
NVRAM
NVRAM
NVRAM
NVRAM

Fig. 1B

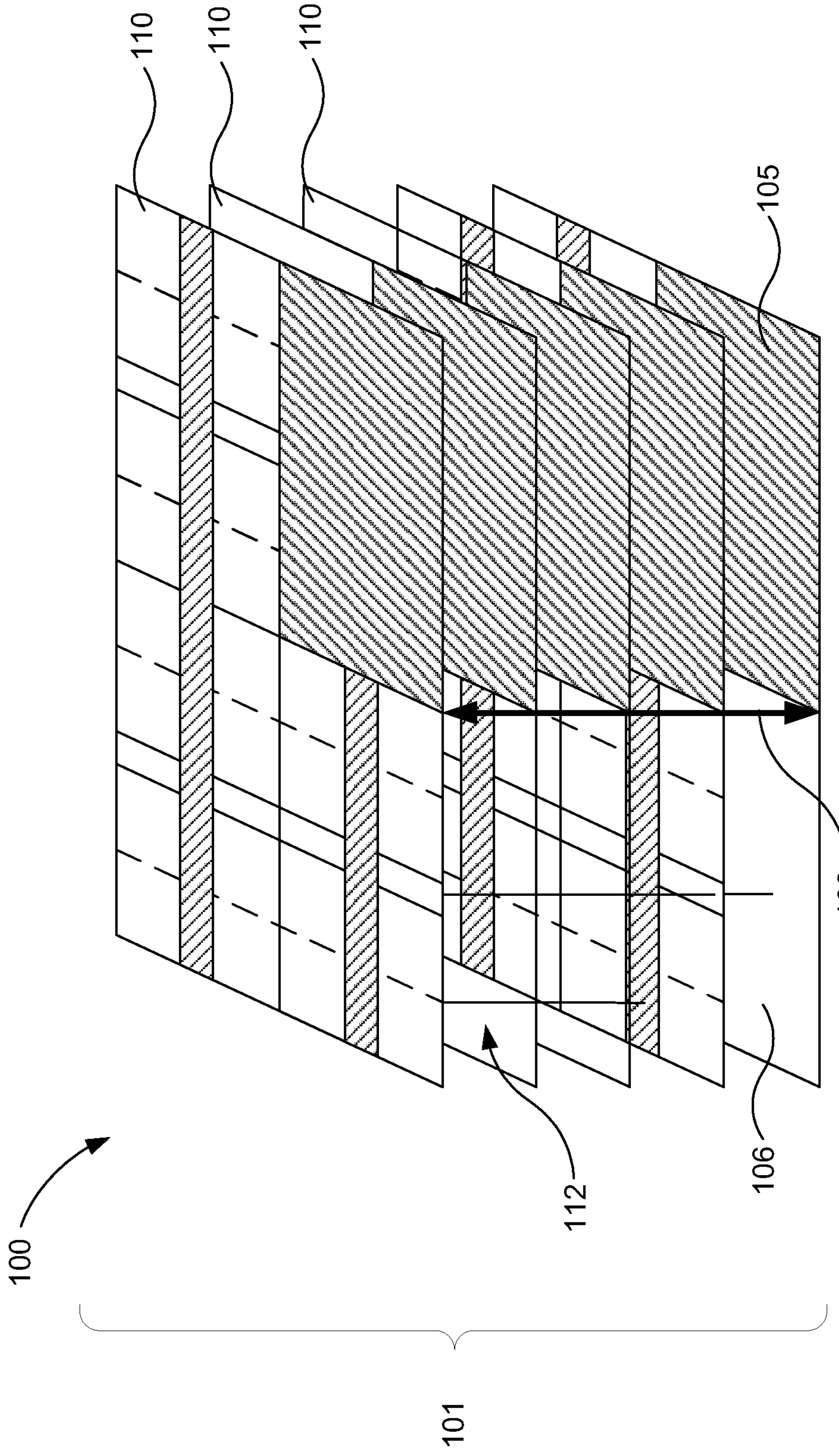

Fig. 1C

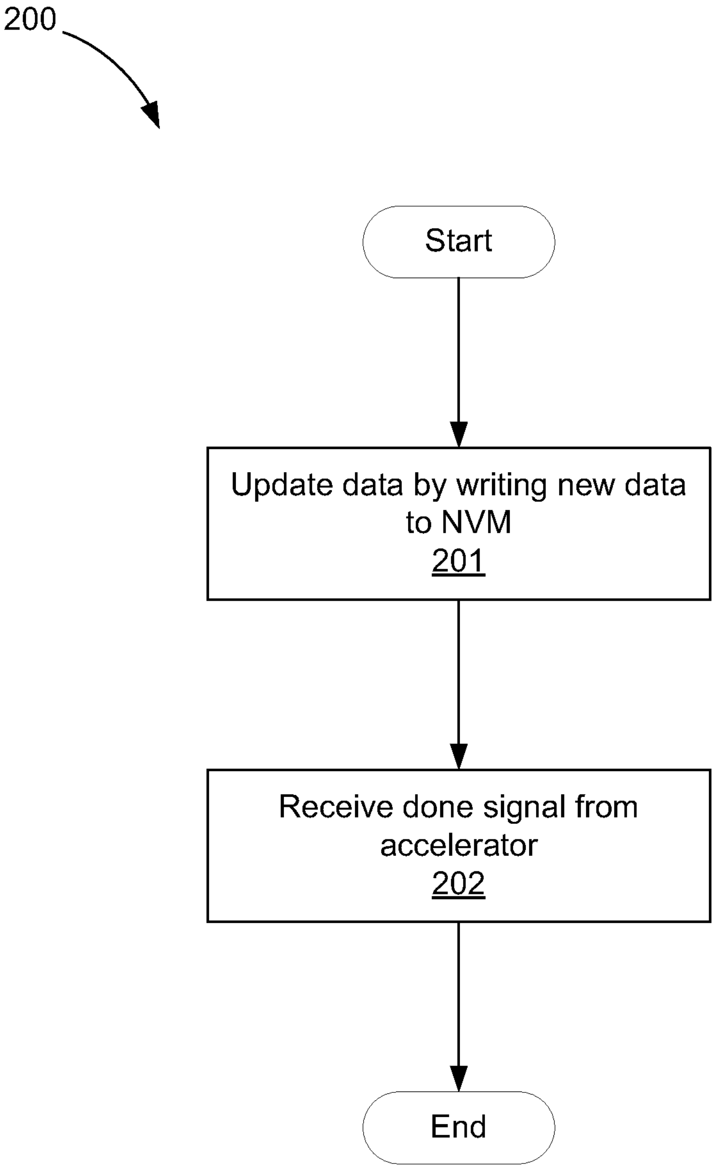
200
Start
Update data by writing new data to NVM
201
Receive done signal from accelerator
202
End

Fig. 2

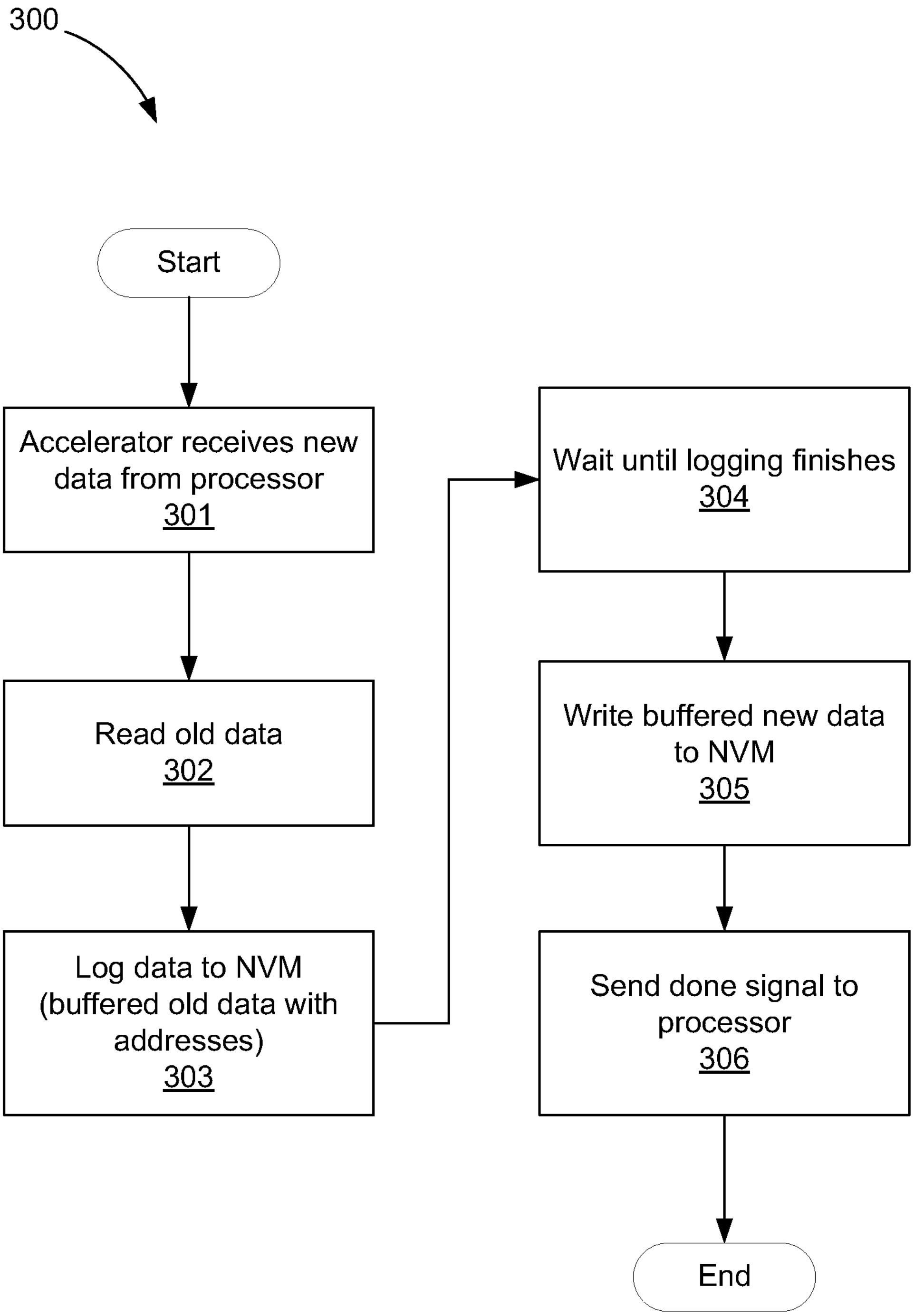
300
Start
Accelerator receives new data from processor 301
Read old data 302
Log data to NVM (buffered old data with addresses) 303
Wait until logging finishes 304
Write buffered new data to NVM 305
Send done signal to processor 306
End

Fig. 3

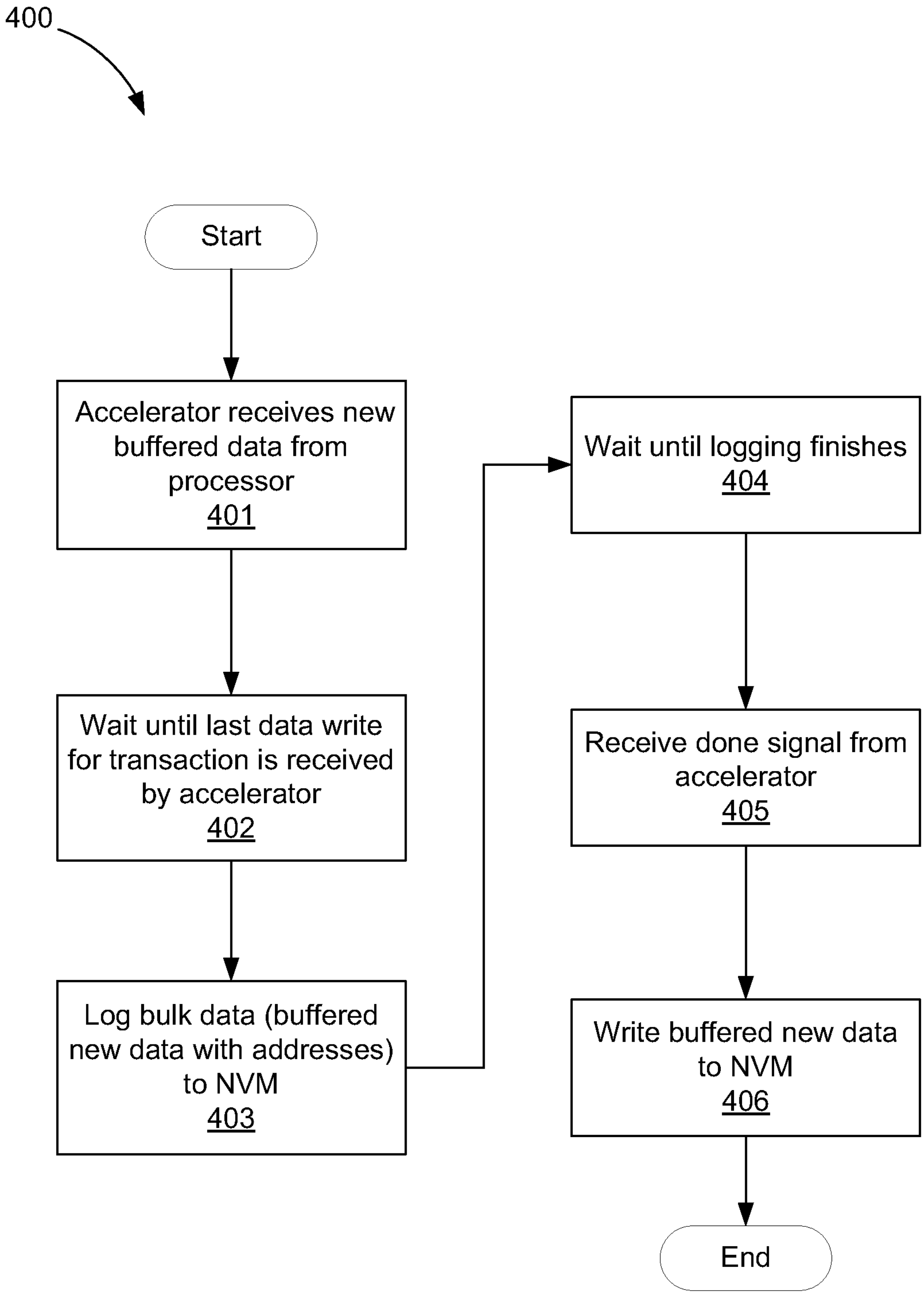
400
Start
Accelerator receives new buffered data from processor
401
Wait until last data write for transaction is received by accelerator
402
Log bulk data (buffered new data with addresses) to NVM
403
Wait until logging finishes
404
Receive done signal from accelerator
405
Write buffered new data to NVM
406
End

Fig. 4

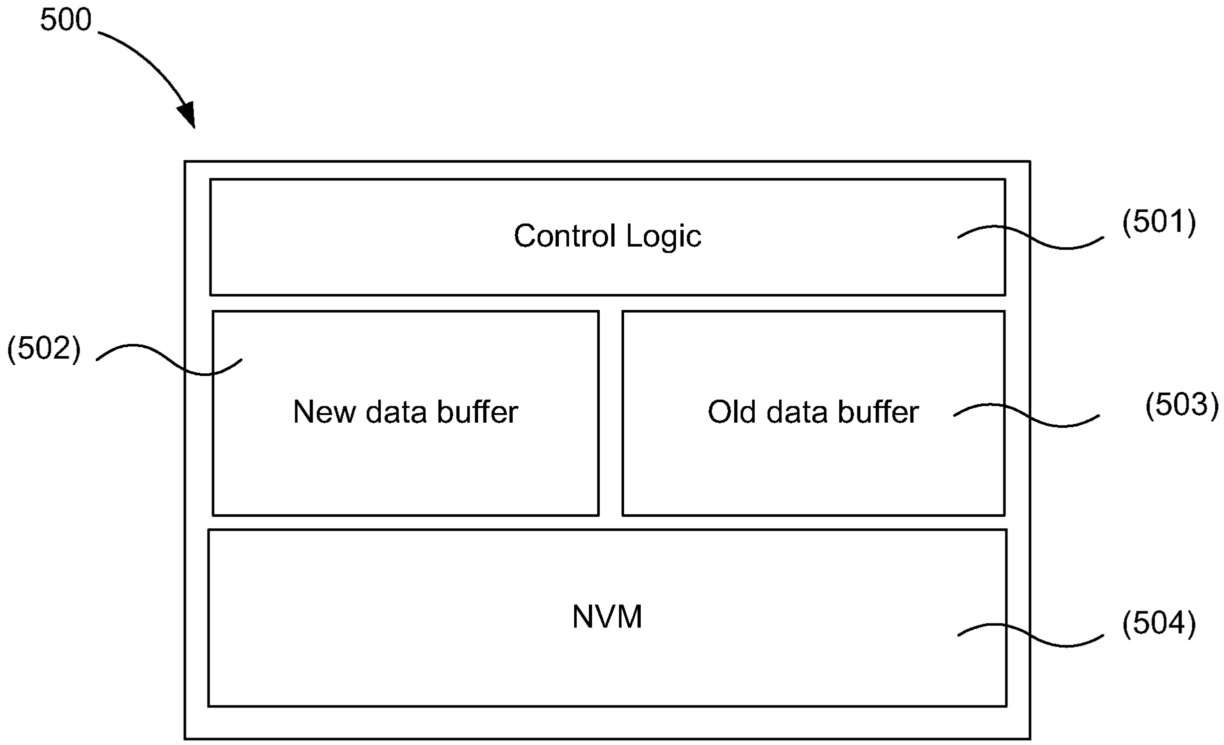
500
Control Logic
(501)
(502)
New data buffer
Old data buffer
(503)
NVM
(504)

Fig. 5A

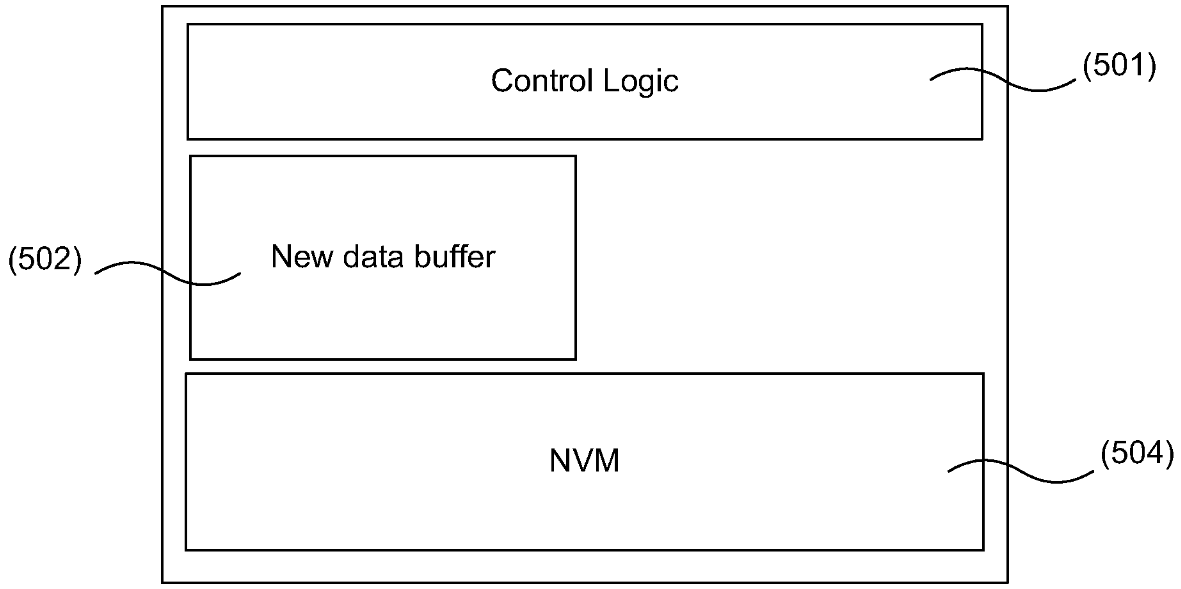
Control Logic
(501)
(502)
New data buffer
NVM
(504)

Fig. 5B

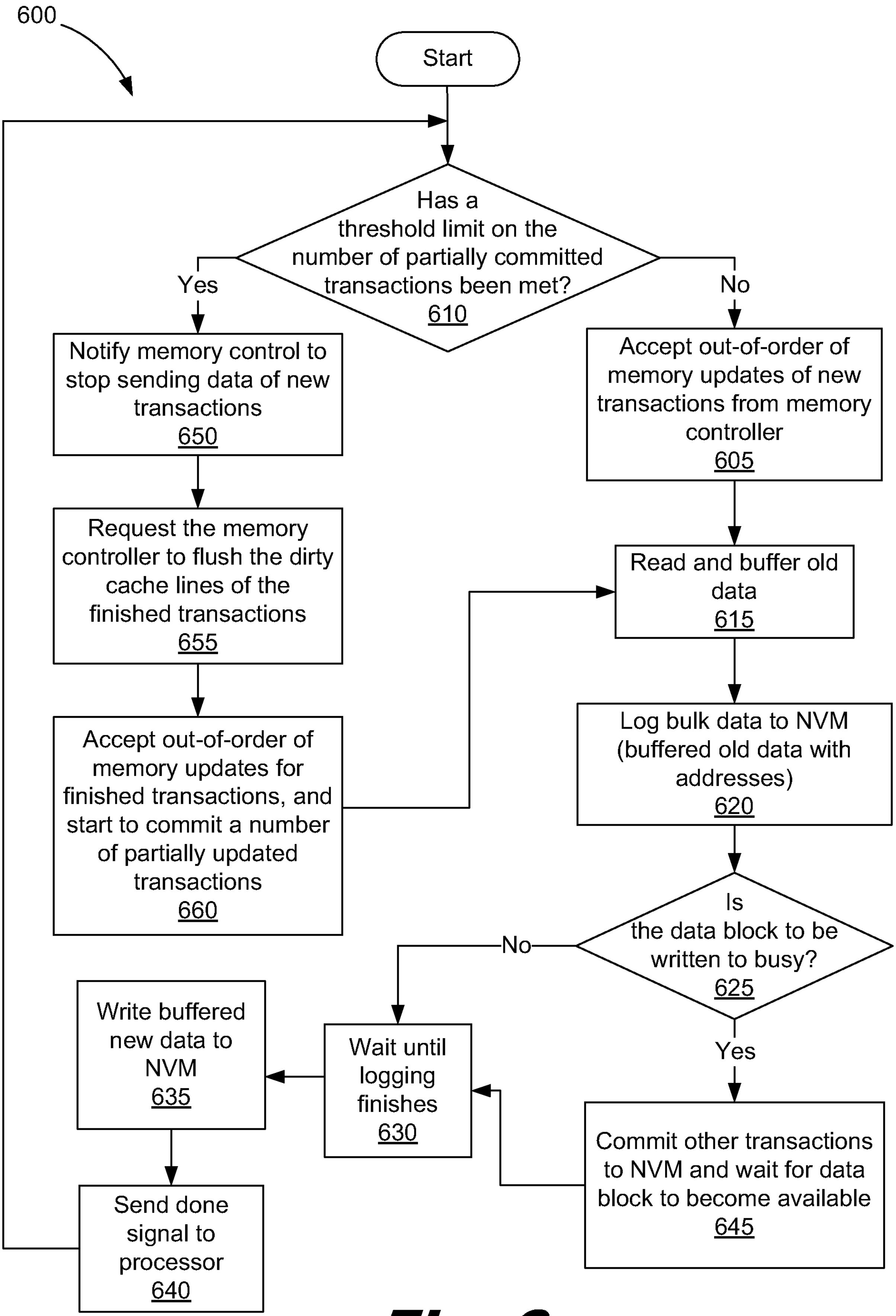
600
Start
Has a threshold limit on the number of partially committed transactions been met?
610
Yes
No
Notify memory control to stop sending data of new transactions
650
Request the memory controller to flush the dirty cache lines of the finished transactions
655
Accept out-of-order of memory updates for finished transactions, and start to commit a number of partially updated transactions
660
Accept out-of-order of memory updates of new transactions from memory controller
605
Read and buffer old data
615
Log bulk data to NVM (buffered old data with addresses)
620
Is the data block to be written to busy?
625
No
Yes
Wait until logging finishes
630
Write buffered new data to NVM
635
Send done signal to processor
640
Commit other transactions to NVM and wait for data block to become available
645

Fig. 6

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US2012/052684

A. CLASSIFICATION OF SUBJECT MATTER

G06F 13/14(2006.01)i, G11C 16/06(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 13/14; G06F 11/07; G06F 12/16; G06F 11/14; G06F 12/00; G06F 13/00

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Korean utility models and applications for utility models
Japanese utility models and applications for utility models

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

eKOMPASS(KIPO internal) & keywords: NVM, power, fail, checkpoint and similar terms.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X A	US 2011-0113208 A1 (NORMAN PAUL JOUPPI et al.) 12 May 2011 See paragraphs 17, 23, 32, 43; and figures 1-4.	1,6-7,10,13,15 2-5,8-9,11-12,14
A	US 2011-0276827 A1 (ROBERT J. ROYER et al.) 10 November 2011 See paragraphs 15-27; and figures 1-4.	1-15
A	US 2008-0059834 A1 (DAVID OWEN ERSTAD) 06 March 2008 See paragraphs 41-44; and figure 4.	1-15
A	US 7516267 B2 (RICHARD L. COULSON et al.) 07 April 2009 See column 7, line 23 - column 8, line 6; and figures 3A-9.	1-15
A	US 5682517 A (THOMAS A. D`ANDREA et al.) 28 October 1997 See column 3, line 38 - column 4, line 23; and figures 2-3C.	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 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 28 March 2013 (28.03.2013)	Date of mailing of the international search report **29 March 2013 (29.03.2013)**
Name and mailing address of the ISA/KR 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 변성철인 Telephone No. 82-42-481-8262

INTERNATIONAL SEARCH REPORT

Information on patent family members

International application No.

PCT/US2012/052684

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 2011-0113208 A1	12.05.2011	CN 102016808 A	13.04.2011
		EP 2271987 A1	12.01.2011
		EP 2271987 A4	20.04.2011
		JP 2011-519460 A	07.07.2011
		KR 10-2011-0002064 A	06.01.2011
		WO 2009-134264 A1	05.11.2009
US 2011-0276827 A1	10.11.2011	US 2010-169710 A1	01.07.2010
		US 7925925 B2	12.04.2011
		US 8312326 B2	13.11.2012
US 2008-0059834 A1	06.03.2008	US 2004-0006723 A1	08.01.2004
		US 2007-0022316 A1	25.01.2007
		US 7058849 B2	06.06.2006
		US 7272747 B2	18.09.2007
		US 7702949 B2	20.04.2010
US 7516267 B2	07.04.2009	CN 101300554 A	05.11.2008
		US 2007-0168698 A1	19.07.2007
		WO 2007-056106 A2	18.05.2007
		WO 2007-056106 A3	08.11.2007
US 05682517 A	28.10.1997	CA 2152204 A1	22.12.1995
		CA 2152204 C	01.09.1998
		DE 69522155 D1	20.09.2001
		DE 69522155 T2	25.04.2002
		EP 0690380 A1	03.01.1996
		EP 0690380 B1	16.08.2001
		US 6027481 A	22.02.2000