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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
07 May 2020 (07.05.2020)

WIPO | PCT

(10) International Publication Number
WO 2020/087265 A1

(51) International Patent Classification:
G06F 3/06 (2006.01) *G06F 12/02* (2006.01)

(21) International Application Number:
PCT/CN2018/112638

(22) International Filing Date:
30 October 2018 (30.10.2018)

(25) Filing Language: English

(26) Publication Language: English

(71) Applicant: **ALIBABA GROUP HOLDING LIMITED** [—/CN]; Fourth Floor, One Capital Place, P.O. Box 847, George Town, Grand Cayman (KY).

(72) Inventors: **ZHOU, Ping**; 400 S El Camino Real, Suite 400, San Mateo, California 94402 (US). **DU, Yu**; Building 1, No.969 West Wen Yi Road, Yu Hang District, Hangzhou, Zhejiang 311121 (CN). **LI, Shu**; 19619 25th Dr. SE, Bothell, Washington 98012 (US).

(74) Agent: **BEIJING TSINGYUANHUI INTELLECTUAL PROPERTY LAW FIRM**; Room 362, 3rd Floor, Suzhou Street No.1, Haidian District, Beijing 100080 (CN).

(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, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

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

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

(54) Title: SYSTEM AND METHOD FOR REPORTING AND HANDLING FLASH PROGRAMMING FAILURE IN HOST-MANAGED FLASH TRANSLATION LAYER

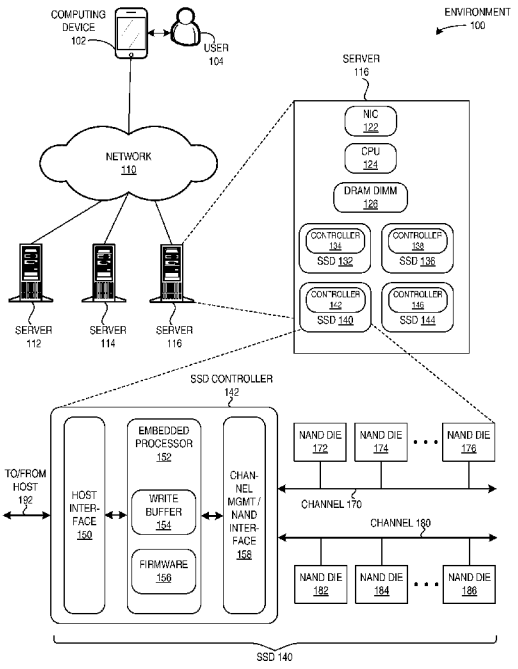


FIG. 1

(57) Abstract: A computer-implemented method for facilitating handling an I/O failure is disclosed. During operation, the system receives, by a controller from a host which includes a flash-translation-layer component, a request to write a plurality of pages to a non-volatile memory device, wherein data is written sequentially in a page stripe which includes physical pages. The system generates programming status information which includes: a status which indicates a success or a failure of whether data in the current page stripe is successfully written to the non-volatile memory device; a first physical block address associated with the status; and a first number of pages which are to be padded in the current page stripe. In response to sending the programming status information to the host, the system copies valid data from the current page stripe to a new physical block address determined by the flash-translation-layer component of the host.

WO 2020/087265 A1

SYSTEM AND METHOD FOR REPORTING AND HANDLING FLASH PROGRAMMING FAILURE IN HOST-MANAGED FLASH TRANSLATION LAYER

Inventors: Ping Zhou, Yu Du, and Shu Li

BACKGROUND

Field

[0001] This disclosure is generally related to the field of data storage. More specifically, this disclosure is related to a system and method for reporting and handling flash programming failure in a host-managed flash translation layer.

Related Art

[0002] The proliferation of the Internet and e-commerce continues to create a vast amount of digital content. Various storage systems and servers have been created to access and store such digital content. A storage system or server can include volatile memory (e.g., dynamic random access memory (DRAM) and multiple drives (e.g., a solid state drive (SSD)). A drive can include non-volatile memory for persistent storage (e.g., NAND flash or flash memory). The memory

in a server plays a crucial role in the performance and capacity of a storage system.

[0003] Flash memory in an SSD is organized into channels/dies. A channel can include multiple dies; a die can include multiple blocks; and a block can include multiple pages. A host typically writes data to an SSD, organized into parity groups. Upon receiving a write request, the SSD controller typically stores the data sequentially into "page stripes," which are physical pages across multiple dies. One of the physical pages in a page stripe is typically used to store parity information. Organizing and writing data across the multiple dies can maximize bandwidth and ensure reliability.

[0004] Because the latency involved in programming flash memory can be high (e.g., on the order of milliseconds), a conventional SSD controller typically returns a write completion to the host as soon as the command is buffered in an internal power-loss protected buffer of the SSD controller. Data is subsequently asynchronously programmed to the flash memory by the SSD controller. While this asynchronous completion can reduce write latency from the perspective of the host, it also introduces challenges in how to handle a failure in programming the data to the flash memory. From the perspective of the host, a write command always succeeds. Thus, in the event of a flash programming failure, the write command has already been acknowledged to and treated as successful by the host. In a conventional SSD controller, a flash translation layer (FTL) component in the SSD controller itself can handle the programming failure.

[0005] However, in an open-channel SSD, where the FTL component resides in the host (e.g., a host-managed FTL) and not on the SSD itself, the host-managed FTL must be notified of a programming failure in both a prompt and a deterministic manner in order to handle the programming failure (e.g., to handle error recovery, data migration, bad block management, and data padding). There

is currently no mechanism which allows a host-managed FTL in an open-channel SSD to properly handle a flash programming failure in a prompt and deterministic manner. This limitation can reduce the flexibility of host software, and may create a bottleneck in the performance of the server. Furthermore, this limitation can result in inefficiencies in the storage system.

SUMMARY

[0006] One embodiment facilitates handling an input/output (I/O) failure. During operation, the system receives, by a controller from a host which includes a flash-translation-layer component, a request to write a plurality of pages to a non-volatile memory device, wherein data is written sequentially in a page stripe which includes physical pages, and wherein each physical page in the page stripe corresponds to a sequentially ordered die of the non-volatile memory device. The system generates programming status information which includes: a status which indicates a success or a failure of whether data in the current page stripe is successfully written to the non-volatile memory device; a first physical block address associated with the status; and a first number of pages which are to be padded in the current page stripe to ensure that the data in the current page stripe can be successfully written to the non-volatile memory device. In response to sending the programming status information to the host, the system copies valid data from the current page stripe to a new physical block address determined by the flash-translation-layer component of the host, thereby facilitating execution of continued asynchronous write requests while allowing the host-based flash-translation-layer component to manage programming failures in the non-volatile memory device.

[0007] In some embodiments, determining that the next available die is the last die of the current page stripe further comprises: writing parity information

associated with the valid data in the current page stripe to a next available page in the last die of the current page stripe.

[0008] In some embodiments, in response to sending the programming status information to the host, and in response to determining that the status indicates a failure, the system: pads the current page stripe based on the first number of pages; determines, by the flash-translation-layer component of the host, the new physical block address to which to write the valid data from the current page stripe; and writes the valid data from the current page stripe to the new physical block address.

[0009] In some embodiments, the new physical block address corresponds to a location in a new page stripe.

[0010] In some embodiments, prior to generating the programming status information, the method further comprises: receiving, by the controller from the host, a first command to obtain the programming status information for the current page stripe, wherein sending the programming status information to the host is in response to receiving the first command.

[0011] In some embodiments, prior to copying the valid data from the current page stripe to the new physical block address, the method further comprises: determining the valid data based on the first physical block address associated with the status.

[0012] In some embodiments, the non-volatile memory device does not include a flash-translation-layer component.

[0013] In some embodiments, the first physical block address and the first number of pages are included in the programming status information only if the status indicates a failure.

BRIEF DESCRIPTION OF THE FIGURES

[0014] FIG. 1 illustrates an exemplary environment that facilitates handling an I/O failure, in accordance with an embodiment of the present application.

[0015] FIG. 2 illustrates an exemplary environment for facilitating handling an I/O failure, in accordance with the prior art.

[0016] FIG. 3A illustrates an exemplary communication for facilitating handling an I/O failure, including a typical host write, in accordance with the prior art.

[0017] FIG. 3B illustrates an exemplary communication for facilitating handling an I/O failure, including a typical host write, in accordance with the prior art.

[0018] FIG. 4 illustrates an exemplary communication for facilitating handling an I/O failure, including a host write in a host-managed flash translation layer, in accordance with an embodiment of the present application.

[0019] FIG. 5 illustrates an exemplary environment for facilitating handling an I/O failure, in accordance with an embodiment of the present application.

[0020] FIG. 6A presents a flowchart illustrating a method for facilitating handling an I/O failure, in accordance with an embodiment of the present application.

[0021] FIG. 6B presents a flowchart illustrating a method for facilitating handling an I/O failure, in accordance with an embodiment of the present application.

[0022] FIG. 7 illustrates an exemplary computer system that facilitates handling an I/O failure, in accordance with an embodiment of the present application.

[0023] FIG. 8 illustrates an exemplary apparatus that facilitates handling an I/O failure, in accordance with an embodiment of the present application.

[0024] In the figures, like reference numerals refer to the same figure elements.

DETAILED DESCRIPTION

[0025] The following description is presented to enable any person skilled in the art to make and use the embodiments, and is provided in the context of a particular application and its requirements. 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 embodiments described herein are not limited to the embodiments shown, but are to be accorded the widest scope consistent with the principles and features disclosed herein.

Overview

[0026] The embodiments described herein provide a system which solves the problem of allowing a host-based FTL in an open-channel SSD to handle a flash programming failure while continuing to execute asynchronous write commands.

[0027] In a host write request, data is typically written sequentially by the SSD controller in a page stripe which includes physical pages, where each physical page in the page stripe corresponds to a sequentially ordered die of the

flash memory. One of the physical pages in the page stripe is used to store parity information. Organizing and writing data across the multiple dies can maximize bandwidth and ensure reliability.

[0028] Because the latency involved in programming flash memory can be high (e.g., on the order of milliseconds), a conventional SSD controller typically returns a write completion to the host as soon as the command is buffered in an internal power-loss protected buffer of the SSD controller. Data is subsequently asynchronously programmed to the flash memory by the SSD controller. While this asynchronous completion can reduce write latency from the perspective of the host, it also introduces challenges in how to handle a failure in programming the data to the flash memory. From the perspective of the host, a write command always succeeds. Thus, in the event of a flash programming failure, the write command has already been acknowledged to and treated as successful by the host. In a conventional SSD controller, a flash translation layer (FTL) component in the SSD controller itself can handle the programming failure.

[0029] However, in an open-channel SSD, where the FTL component resides in the host (e.g., a host-managed FTL) and not on the SSD itself, the host-managed FTL must be notified of a programming failure in both a prompt and a deterministic manner in order to handle the programming failure (e.g., to handle error recovery, data migration, bad block management, and data padding). There is currently no mechanism which allows a host-managed FTL in an open-channel SSD to properly handle a flash programming failure in a prompt and deterministic manner. This limitation can reduce the flexibility of host software, and may create a bottleneck in the performance of the server. Furthermore, this limitation can result in inefficiencies in the storage system.

[0030] One solution, as proposed in the Open-Channel Specification 2.0, is to make all write commands synchronous, such that a write completion always

returns the true status of flash programming. However, this solution dramatically increases the write latency from the perspective of the host, from an order of microseconds to an order of milliseconds. Handling such a significant increase to the write latency may require a significant change in the design of the host software, which is an impractical solution for many legacy solutions which are already widely deployed online.

[0031] Another solution is to report a programming failure event through a Non-Volatile Memory (NVMe) Asynchronous Event mechanism. However, this mechanism is neither prompt nor reliable, because it relies on interrupts and does not guarantee delivery of notifications. Multiple events may be coalesced, which can cause a significant loss of information. Furthermore, this mechanism is not deterministic, which does not allow the host any visibility or predictability into when it may receive any notifications via this mechanism.

[0032] The embodiments described herein address these limitations by creating a new synchronous command: the "Parity End" command. Write commands from the host are still written asynchronously, thus maintaining the short write latency from the perspective of the host, as in the prior art. By using the Parity End command, the host can substantially synchronously command the controller to generate and write parity data for the current page stripe, and also return programming status information of the current page stripe. The programming status information can include: 1) a status which indicates a success or a failure of whether data in the current page stripe has been successfully written or programmed to the flash memory; 2) a physical block address associated with the status, if the status indicates a failure; and 3) a number of additional pages or portions of pages to pad to ensure that the data in the current page stripe is written or programmed successfully to the flash memory, if the status indicates a failure. Upon receiving the programming status information, if the status indicates a

failure, the host-based FTL can pad the current page stripe based on the number of pages, determine a new physical block address to which to write valid data, and write the valid data to the new physical block address. An exemplary communication between a host, an SSD controller, and a NAND channel is described below in relation to FIGs. 4 and 5.

[0033] Thus, the embodiments described herein provide a system which improves the efficiency and performance of a storage system. The system provides a mechanism by which a host-based FTL (e.g., as in an open-channel SSD) can be notified of a (reported) programming failure, and can subsequently handle the reported programming failure based on information in the synchronous Parity End command, while maintaining the asynchronous nature of write commands from the host. That is, the system need not sacrifice the write latency from the perspective of the host in order to handle the flash programming failures in an open-channel SSD. Furthermore, the embodiments described herein provide a technological solution (introducing the Parity End command, which generates and reports the programming information status) to a technological problem in the software arts (maintaining the execution of asynchronous write requests while allowing the host-based FTL, as in an open-channel SSD, to handle programming failures to the flash memory).

[0034] The terms "flash translation layer component," "flash-translation-layer component," "FTL," and "FTL component" are used interchangeably in this disclosure, and refer to a layer or component which maps host side or file system logical block addresses (LBAs) to the physical block addresses (PBAs) of flash memory (e.g., the logical-to-physical mapping). In a conventional SSD, the FTL component is typically included in the SSD controller. However, in an open-channel SSD, the FTL component is a host-based FTL and resides in the host.

That is, the host-based FTL component is responsible for performing the logical-to-physical mapping.

[0035] The terms "host-managed FTL" and "host-based FTL" are used interchangeably in this disclosure, and refer to an FTL component which is managed by the host. That is, instead of residing in the SSD controller or on the storage device, the FTL component resides in the host (as in an open-channel SSD).

Exemplary Environment and Network

[0036] FIG. 1 illustrates an exemplary environment 100 that facilitates handling an I/O failure, in accordance with an embodiment of the present application. Environment 100 can include a computing device 102 and an associated user 104. Computing device 102 can communicate via a network 110 with storage servers 112, 114, and 116, which can be part of a distributed storage system and accessed via client servers (not shown). A storage server can include multiple storage drives, and each drive can include a controller and multiple physical media for data storage. For example, server 116 can include a network interface card (NIC) 122, a central processing unit (CPU) 124, a dynamic random access memory dual in-line memory module (DRAM DIMM) 126, and SSDs 132, 136, 140, and 144 with, respectively, controllers 134, 138, 142, and 146.

[0037] A controller can include interfaces to a host and to a non-volatile memory. A controller can also include a write buffer, which is power-loss protected, as well as firmware which includes instructions and/or code to execute the methods described herein. For example, SSD 140 can include SSD controller 142. Controller 142 can include: a host interface 150; an embedded processor 152, which includes a write buffer 154 and a firmware 156; and a channel management 158. SSD controller 142 can communicate with a host (e.g., via host interface

150 and a communication to/from host 192). SSD controller 142 can also communicate with the non-volatile memory (via channel management 158). The non-volatile memory can be accessed via multiple channels. For example, NAND dies 172, 174, and 176 may be accessed via a channel 170, and NAND dies 182, 184, and 186 may be accessed via a channel 180.

[0038] During operation, in the embodiments described herein, firmware 156 can include instructions and/or code which include generating a "Parity End" command, which, upon reaching the end of a page stripe, causes the controller to return to the host: 1) the programming status of the page stripe (e.g., a success or a failure); 2) a physical location (such as a physical block address) associated with a failure status; and 3) a number of pages which are to padded in the page stripe in order to ensure that the data in the page stripe can be successfully written to the non-volatile memory. The Parity End command and related communications are described below in relation to FIGs. 4 and 5.

Exemplary Environment and Communication in the Prior Art

[0039] FIG. 2 illustrates an exemplary environment 200 for facilitating handling an I/O failure, in accordance with the prior art. Environment 200 can include a host 202, which performs a host write 204 request. Environment 200 includes a non-volatile memory, which can include multiple dies, such as dies 208, 218, 228, and 238. Each die can include multiple blocks, and each block can include multiple pages. For example, die 208 can include blocks 210, 212, 214, and 216, and block 212 can include a physical page 211. Similarly, blocks 222, 232, and 242 can include, respectively, physical pages 221, 231, and 241. In environment 200, a diagonally shaded pattern indicates that (relevant) data has been stored or filled in the respective unit, while a clear pattern (i.e., no pattern) indicates that data has not yet been stored or filled in the respective unit.

[0040] Recall that a page stripe can include physical pages across multiple dies. In environment 200, blocks 210, 220, 230, and 240 are completely filled in with data (as indicated with the diagonally shaded pattern). Physical pages 211, 221, 231, and 241 (of, respectively, blocks 212, 222, 232, and 242) are partially filled in (as indicated by the diagonally shaded pattern in 211.1, 221.1, 231.1, and 241.1), and comprise a page stripe 209.

[0041] During operation, in executing host write 204, the conventional system can write data to the pages of page stripe 209 in a "horizontal" manner, e.g., as 4 KB or 4K pages. That is, the system can write 4K of data to the next available portion (211.1) of a current physical page (211) of a first sequentially ordered die (208) in the page stripe. Then system can then continue to write additional 4K segments to the next available portion (221.1) of the next current physical page (221) of the next sequentially ordered die (218), write to the next available portion (231.1) of the next current physical page (231), and finally write the parity information to the next available portion (241.1) of the parity page (241).

[0042] The conventional system can continue to write data in a horizontal page stripe manner. However, if a programming failure 250 occurs at page portion 221.2, the SSD controller may continue to write data to the non-volatile memory even after failure 250 has occurred. In the conventional SSD (where the FTL is in the SSD controller), failure 250 is handled internally by the SSD controller, and thus is transparent to the host. However, in the conventional open-channel SSD (where the FTL is in the host), the system lacks a mechanism to report the failure to the host, and there is also no way for the host to handle the failure. The host-based FTL needs to know about the program failure in order to handle error recovery, data migration, bad block management, and data padding.

Thus, the conventional open-channel SSDs do not allow the host-based FTL to properly handle such programming failures.

[0043] FIG. 3A illustrates an exemplary communication 300 for facilitating handling an I/O failure, including a typical successful host write, in accordance with the prior art. Communication 300 can include communications between a host 302, an SSD controller 304, and a NAND channel 306. Host 302 can send a series of write commands to SSD controller 304 and receive a series of corresponding write completions from SSD controller 304. The corresponding programming command sent from SSD controller 304 to NAND channel 306 is performed asynchronously. For example, host 302 can send a write command 312 to and receive a write completion 314 from SSD controller 304. Host 302 can also send a write command 316 to and receive a write completion 318 from SSD controller 304. Host 302 can also send a write command 320 to and receive a write completion 322 from SSD controller 304. At a time which is asynchronous with this series of write commands/completions, SSD controller can send a program command 332 to NAND channel 306, which can perform a NAND programming 340 (e.g., by writing data associated with write commands 312, 316, and 320 to the non-volatile memory). Upon the successful completion of NAND programming 340, NAND channel 306 can send a program completion command 334 to SSD controller 304. Note that in this example, program command 332 is sent by SSD controller 304 to NAND channel 306 subsequent to SSD controller 304 sending write completion 314 back to host 302. Furthermore, program completion 334 is received by SSD controller 304 subsequent to SSD controller sending write completion 322 to host 302.

[0044] FIG. 3B illustrates an exemplary communication 350 for facilitating handling an I/O failure, including a typical unsuccessful host write, in accordance with the prior art. Communication 350 is similar to communication

300 of FIG. 3A, with the exception that NAND programming 340 yields an unsuccessful result (e.g., as depicted above in relation to failure 250 of FIG. 2). In this case, NAND channel 306 can send a program failure 352 to SSD controller 304 (e.g., at a failure 354). However, there is no mechanism to report program failure 352 to host 302, nor is there any mechanism for host 302 to handle program failure 352 (or failure 354).

Exemplary Communication and Environment for Facilitating Flash Storage Management Using Parity End Command (Programming Status Information)

[0045] FIG. 4 illustrates an exemplary communication 400 for facilitating handling an I/O failure, including a host write in a host-managed flash translation layer, in accordance with an embodiment of the present application. Communication 400 includes similar communications as in communication 300 of FIG. 3A, but also includes the Parity End command, which can return the status of programming the non-volatile memory, the location of the program failure (if any), and the number of pages (if any) to pad to ensure a safe data migration. At the same time, the system can maintain the asynchronous write and programming commands, which, unlike the existing solutions, does not result in a increased latency, a significant software design change, or a lack of visibility and predictability.

[0046] Specifically, a set of asynchronous write commands can be sent from host 302 to SSD controller 304, and a corresponding set of program commands can be sent from SSD controller 304 to NAND channel 306. Host 302 can also send a Parity End command 422 to SSD controller 304, which can trigger SSD controller 304 to return certain programming status information back to host 302 (via a Parity End completion 424, which includes the programming status information). Host 302 can subsequently use the returned programming status

information to perform, by the host-based FTL, the proper error recovery functions, including data migration, bad block management, and data padding.

[0047] For example, a program command 432 can result in NAND channel 306 executing a NAND programming 436, which can end in a (successful) program completion 434, which successful program completion 434 is sent back to SSD controller 304. Similarly, a program command 442 can result in NAND channel 306 executing a NAND programming 446, which can end in a program failure 444 (or an unsuccessful program completion). SSD controller 304 can subsequently send Parity End completion 424 back to host 302. Parity End completion can include: 1) the programming status of the page stripe (e.g., a success or a failure); 2) a physical location (such as a physical block address) associated with a failure status; and 3) a number of pages which are to padded in the page stripe in order to ensure that the data in the page stripe can be successfully written to the non-volatile memory. This allows the host-based FTL to pad the page stripe based on the number of pages (as included in the Parity End completion), and to determine the valid data in the page stripe based on the physical location associated with the failure status (also as included in the Parity End completion). The host-based FTL can also determine a new physical block address to which to write the valid data from the page stripe. Subsequently, the host-based FTL can write the valid data from the page stripe to the new physical block address.

[0048] Thus, embodiments of the system described herein provide a mechanism which both reports the programming failure to the host, and allows the host to manage the programming failure, without sacrificing the asynchronous nature of the write commands from the host to the SSD controller, and the programming commands from the SSD controller to the NAND channel.

[0049] FIG. 5 illustrates an exemplary environment 500 for facilitating handling an I/O failure, in accordance with an embodiment of the present application. Environment 500 can include a host 502, which performs a host write 552 request and a Parity End command 554. Host 502 can also receive a Parity End completion message 556. Environment 500 includes a non-volatile memory, which can include multiple dies, such as dies 508, 518, 528, and 538. Each die can include multiple blocks, and each block can include multiple pages. For example, die 508 can include blocks 510, 512, 514, and 516, and block 512 can include a physical page 511. Similarly, blocks 522, 532, and 542 can include, respectively, physical pages 521, 531, and 541. In environment 500, a diagonally shaded pattern indicates that (relevant) data has been stored or filled in the respective unit, while a clear pattern (i.e., no pattern) indicates that data has not yet been stored or filled in the respective unit.

[0050] Recall that a page stripe can include physical pages across multiple dies. In environment 500, blocks 510, 520, 530, and 540 are completely filled in with data (as indicated with the diagonally shaded pattern). Physical pages 511, 521, 531, and 541 (of, respectively, blocks 512, 522, 532, and 542) are partially filled in (as indicated by the diagonally shaded pattern in pages 511, 521, 531, and 541, and comprise a page stripe 509.

[0051] During operation, in executing host write 504, the system can write data to the pages of page stripe 509 in a "horizontal" manner (similar to that described above in relation to FIG. 2). Upon reaching the end of the page stripe (e.g., detecting or determining that the next available die is a last die of the current page stripe), the system can execute Parity End command 554 by writing parity information associated with the valid data in page stripe 509, and generating programming status information which includes: 1) the programming status of page stripe 509 (e.g., a success or a failure); 2) a physical location (such as a

physical block address) associated with the status, if the status indicates a failure; and 3) a number of pages (if the status indicates a failure) which are to padded in page stripe 509 in order to ensure that the data in page stripe 509 is successfully written to the non-volatile memory. Note that the system can also pad incomplete physical pages, or can return a portion or fraction of a whole number of physical pages which are to be padded to ensure the safe data migration. The system can return this programming status information back to host 502 as part of the Parity End completion 556 message.

[0052] Thus, in environment 500, the system provides the mechanism to both report the programming failure status to the host and allow the host to handle the error recovery, which mechanism is missing from the conventional open-channel SSD. Specifically, the system of environment 500 allows for the continued execution of asynchronous write requests without sacrificing the write latency from the perspective of the host, by reporting failures in flash programming directly to the host. This allows the host-based FTL (as in an open-channel SSD) to handle the error recovery in a timely and deterministic manner, which improves the efficiency of the storage system.

Method for Facilitating Flash Storage Management

[0053] FIG. 6A presents a flowchart 600 illustrating a method for facilitating handling an I/O failure, in accordance with an embodiment of the present application. During operation, the system receives, by a controller from a host which includes a flash-translation-layer component, a request to write a plurality of pages to a non-volatile memory device, wherein data is written sequentially in a page stripe which includes physical pages (operation 602). Each physical page in the page stripe corresponds to a sequentially ordered die of the non-volatile memory device. The system receives, by the controller from the host,

a first command to obtain programming status information for a current page stripe, wherein the programming status information includes: a status which indicates a success or a failure of whether data in the current page stripe is successfully written to the non-volatile memory device; a first physical block address associated with the status, if the status indicates a failure; and a first number of pages (if the status indicates a failure) which are to padded in the current page stripe in order to ensure that the data in the page stripe can be successfully written to the non-volatile memory device (operation 604).

[0054] If the next available die is not the last die of the current page stripe (decision 606), the operation can continue at operation 602 or return (not shown). If the next available die is the last die of the current page stripe (decision 606), the system generates the programming status information for the current page stripe (operation 608). The system writes parity information associated with valid data in the current page stripe to a next available page in the last die of the current page stripe (operation 610). In response to sending the programming status information to the host, the system copies valid data from the current page stripe to a new physical block address determined by the flash-translation-layer component of the host, thereby facilitating execution of continuous asynchronous write requests while allowing the host-based flash-translation-layer component to manage programming failures in the non-volatile memory device (operation 612).

[0055] FIG. 6B presents a flowchart 620 illustrating a method for facilitating handling an I/O failure, in accordance with an embodiment of the present application. The operations of flowchart 620 can be performed by the host-based FTL component, and can occur in response to sending the programming status information to the host. During operation, the system receives, by the host from the controller, the programming status information

(operation 622). If the status included in the programming status information does not indicate a failure (decision 624), the operation returns.

[0056] If the status included in the programming status information does indicate a failure (decision 624), the system pads the current page stripe based on the first number of pages (operation 626). The system determines the valid data from the current page stripe based on the first physical block address associated with the status (operation 628). The system determines, by the flash-translation-layer component of the host, a new physical block address to which to write the valid data from the current page stripe (operation 630). The system writes the valid data from the current page stripe to the new physical block address (operation 632).

Exemplary Computer System and Apparatus

[0057] FIG. 7 illustrates an exemplary computer system 700 that facilitates handling an I/O failure, in accordance with an embodiment of the present application. Computer system 700 includes a processor 702, a memory 704, a non-volatile memory 706, and a storage device/firmware 708. Computer system 500 may be a computing device or a storage device. Volatile memory 704 can include memory (e.g., RAM) that serves as a managed memory, and can be used to store one or more memory pools. Non-volatile memory 706 can include memory (e.g., NAND flash) which is used for persistent storage. Furthermore, computer system 700 can be coupled to a display device 710, a keyboard 712, and a pointing device 714. Storage device/firmware 708 can store an operating system 716, a content-processing system 718, and data 732. Note that firmware 708 may alternatively be located in or included in other components of computer system 700.

[0058] Content-processing system 718 can include instructions, which when executed by computer system 700, can cause computer system 700 to perform methods and/or processes described in this disclosure. For example, content-processing system 718 can include instructions for receiving and transmitting data packets, including a request to write or read data, data to be encoded and stored, a block of data, a page of data, a command, and programming status information.

[0059] Content-processing system 718 can further include instructions for receiving, by a controller from a host which includes a flash-translation-layer component, a request to write a plurality of pages to a non-volatile memory device, wherein data is written sequentially in a page stripe which includes physical pages, and wherein each physical page in the page stripe corresponds to a sequentially ordered die of the non-volatile memory device (communication module 720). Content-processing system 718 can include instructions for determining that a next available die is a last die of the current page stripe (data-writing module 722). Content-processing system 718 can also include instructions for generating programming status information which includes: a status which indicates a success or a failure of whether data in the current page stripe is successfully written to the non-volatile memory device; a first physical block address associated with the status; and a first number of pages which are to be padded in the current page stripe to ensure that the data in the current page stripe can be successfully written to the non-volatile memory device (programming status-generating module 724). Content-processing system 718 can include instructions for, in response to sending the programming status information to the host (communication module 720), copying valid data from the current page stripe to a new physical block address determined by the flash-translation-layer component of the host (data-copying module 726), thereby

facilitating execution of continued asynchronous write requests while allowing the host-based flash-translation-layer component to manage programming failures in the non-volatile memory device (data-writing module 722).

[0060] Content-processing system 718 can additionally include instructions for writing parity information associated with the valid data in the current page stripe to a next available page in the last die of the current page stripe (parity information-managing module 728). Content-processing system 718 can include instructions for, in response to sending the programming status information to the host (communication module 720) and in response to determining that the status indicates a failure (data-copying module 726): padding the current page stripe based on the first number of pages (data-copying module 726); determining, by the flash-translation-layer component of the host, the new physical block address to which to write the valid data from the current page stripe (data-copying module 726); and writing the valid data from the current page stripe to the new physical block address (data-copying module 726).

[0061] Data 732 can include any data that is required as input or that is generated as output by the methods and/or processes described in this disclosure. Specifically, data 732 can store at least: data to be stored, written, loaded, moved, retrieved, deleted, or copied; a logical unit of data; a logical block address (LBA); a physical unit of data; a physical block address (PBA); a physical page of data; a block of data; a page stripe; sequentially written data; a sequentially ordered die; a command; a write command; a write completion; a program command; a program completion; a program failure; programming status information; a status; a status which indicates a success or a failure; a number of pages to page; parity information; a new physical block address; valid data in a page stripe; a location in a new page stripe; a command to obtain programming status information for a

page stripe; an indicator of a programming failure; a Parity End command; a Parity End completion; and a flash-translation-layer component.

[0062] FIG. 8 illustrates an exemplary apparatus 800 that facilitates handling an I/O failure, in accordance with an embodiment of the present application. Apparatus 800 can comprise a plurality of units or apparatuses which may communicate with one another via a wired, wireless, quantum light, or electrical communication channel. Apparatus 800 may be realized using one or more integrated circuits, and may include fewer or more units or apparatuses than those shown in FIG. 8. Further, apparatus 800 may be integrated in a computer system, or realized as a separate device which is capable of communicating with other computer systems and/or devices. Specifically, apparatus 800 can comprise units 802-812 which perform functions or operations similar to modules 720-730 of computer system 700 of FIG. 7, including: a communication unit 802; a data-writing unit 804; a programming status-generating unit 806; a data-copying unit 808; a parity information-managing unit 810; and a command-generating unit 812.

[0063] The data structures and code described in this detailed description are typically stored on a computer-readable storage medium, which may be any device or medium that can store code and/or data for use by a computer system. The computer-readable storage medium includes, but is not limited to, volatile memory, non-volatile memory, magnetic and optical storage devices such as disk drives, magnetic tape, CDs (compact discs), DVDs (digital versatile discs or digital video discs), or other media capable of storing computer-readable media now known or later developed.

[0064] The methods and processes described in the detailed description section can be embodied as code and/or data, which can be stored in a computer-readable storage medium as described above. When a computer system reads and executes the code and/or data stored on the computer-readable storage medium,

the computer system performs the methods and processes embodied as data structures and code and stored within the computer-readable storage medium.

[0065] Furthermore, the methods and processes described above can be included in hardware modules. For example, the hardware modules can include, but are not limited to, application-specific integrated circuit (ASIC) chips, field-programmable gate arrays (FPGAs), and other programmable-logic devices now known or later developed. When the hardware modules are activated, the hardware modules perform the methods and processes included within the hardware modules.

[0066] The foregoing embodiments described herein have been presented for purposes of illustration and description only. They are not intended to be exhaustive or to limit the embodiments described herein 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 embodiments described herein. The scope of the embodiments described herein is defined by the appended claims.

What Is Claimed Is:

1. A computer-implemented method for facilitating handling an I/O failure, the method comprising:

receiving, by a controller from a host which includes a flash-translation-layer component, a request to write a plurality of pages to a non-volatile memory device, wherein data is written sequentially in a page stripe which includes physical pages, and wherein each physical page in the page stripe corresponds to a sequentially ordered die of the non-volatile memory device;

generating programming status information which includes:

a status which indicates a success or a failure of whether data in the current page stripe is successfully written to the non-volatile memory device;

a first physical block address associated with the status; and

a first number of pages which are to be padded in the current page stripe to ensure that the data in the current page stripe can be successfully written to the non-volatile memory device; and

in response to sending the programming status information to the host, copying valid data from the current page stripe to a new physical block address determined by the flash-translation-layer component of the host.

2. The method of claim 1, further comprising determining that a next available die is a last die of the current page stripe, which involves:

writing parity information associated with the valid data in the current page stripe to a next available page in the last die of the current page stripe.

3. The method of claim 1, wherein in response to sending the

programming status information to the host, the method further comprises:

in response to determining that the status indicates a failure:

padding the current page stripe based on the first number of pages;

determining, by the flash-translation-layer component of the host, the new physical block address to which to write the valid data from the current page stripe; and

writing the valid data from the current page stripe to the new physical block address.

4. The method of claim 3, wherein the new physical block address corresponds to a location in a new page stripe.

5. The method of claim 1, wherein prior to generating the programming status information, the method further comprises:

receiving, by the controller from the host, a first command to obtain the programming status information for the current page stripe,

wherein sending the programming status information to the host is in response to receiving the first command.

6. The method of claim 1, wherein prior to copying the valid data from the current page stripe to the new physical block address, the method further comprises:

determining the valid data based on the first physical block address associated with the status.

7. The method of claim 1, wherein the non-volatile memory device does not include a flash-translation-layer component.

8. The method of claim 1, wherein the first physical block address and the first number of pages are included in the programming status information only if the status indicates a failure.

9. A computer system for facilitating handling an I/O failure, the system comprising:

a processor; and

a memory coupled to the processor and storing instructions, which when executed by the processor cause the processor to perform a method, the method comprising:

receiving, by a controller from a host which includes a flash-translation-layer component, a request to write a plurality of pages to a non-volatile memory device, wherein data is written sequentially in a page stripe which includes physical pages, and wherein each physical page in the page stripe corresponds to a sequentially ordered die of the non-volatile memory device;

generating programming status information which includes:

a status which indicates a success or a failure of whether data in the current page stripe is successfully written to the non-volatile memory device;

a first physical block address associated with the status; and

a first number of pages which are to be padded in the current page stripe to ensure that the data in the current page stripe can be successfully written to the non-volatile memory device; and

in response to sending the programming status information to the host, copying valid data from the current page stripe to a new physical block address determined by the flash-translation-layer component of the host.

10. The computer system of claim 9, wherein the method further comprises determining that a next available die is a last die of the current page stripe, which involves:

writing parity information associated with the valid data in the current page stripe to a next available page in the last die of the current page stripe.

11. The computer system of claim 9, wherein in response to sending the programming status information to the host, the method further comprises:

in response to determining that the status indicates a failure:

padding the current page stripe based on the first number of pages;

determining, by the flash-translation-layer component of the host, the new physical block address to which to write the valid data from the current page stripe; and

writing the valid data from the current page stripe to the new physical block address.

12. The computer system of claim 11, wherein the new physical block address corresponds to a location in a new page stripe.

13. The computer system of claim 9, wherein prior to generating the programming status information, the method further comprises:

receiving, by the controller from the host, a first command to obtain the programming status information for the current page stripe,

wherein sending the programming status information to the host is in response to receiving the first command.

14. The computer system of claim 9, wherein prior to copying the

valid data from the current page stripe to the new physical block address, the method further comprises:

determining the valid data based on the first physical block address associated with the status.

15. The computer system of claim 1, wherein the non-volatile memory device does not include a flash-translation-layer component.

16. The computer system of claim 9, wherein the first physical block address and the first number of pages are included in the programming status information only if the status indicates a failure.

17. An apparatus for facilitating handling an I/O failure, the apparatus comprising:

a communication unit configured to receive, by a controller from a host which includes a flash-translation-layer component, a request to write a plurality of pages to a non-volatile memory device, wherein data is written sequentially in a page stripe which includes physical pages, and wherein each physical page in the page stripe corresponds to a sequentially ordered die of the non-volatile memory device;

a programming status-generating unit configured to generate programming status information which includes:

a status which indicates a success or a failure of whether data in the current page stripe is successfully written to the non-volatile memory device;

a first physical block address associated with the status; and

a first number of pages which are to be padded in the current page

stripe to ensure that the data in the current page stripe can be successfully written to the non-volatile memory device; and

a data-copying unit configured to, in response to the communication unit sending the programming status information to the host, copy valid data from the current page stripe to a new physical block address determined by the flash-translation-layer component of the host,

thereby facilitating execution of continued asynchronous write requests while allowing the host-based flash-translation-layer component to manage programming failures in the non-volatile memory device.

18. The apparatus of claim 17, further comprising:

a data-writing unit configured to determine that a next available die is a last die of the current page stripe; and

a parity information-managing unit configured to write parity information associated with the valid data in the current page stripe to a next available page in the last die of the current page stripe.

19. The apparatus of claim 17, wherein in response to the communication unit sending the programming status information to the host, the data-copying unit is further configured to:

in response to determining that the status indicates a failure:

pad the current page stripe based on the first number of pages;

determine, by the flash-translation-layer component of the host, the new physical block address to which to write the valid data from the current page stripe; and

write the valid data from the current page stripe to the new physical block address.

20. The apparatus of claim 17, wherein prior to the programming status-generating unit generating the programming status information, the apparatus further comprises:

a command-generating unit configured to generated a first command to obtain the programming status information for the current page stripe,

wherein the communication unit is further configured to:

receive, by the controller from the host, the first command, and

send the programming status information to the host is in response to receiving the first command.

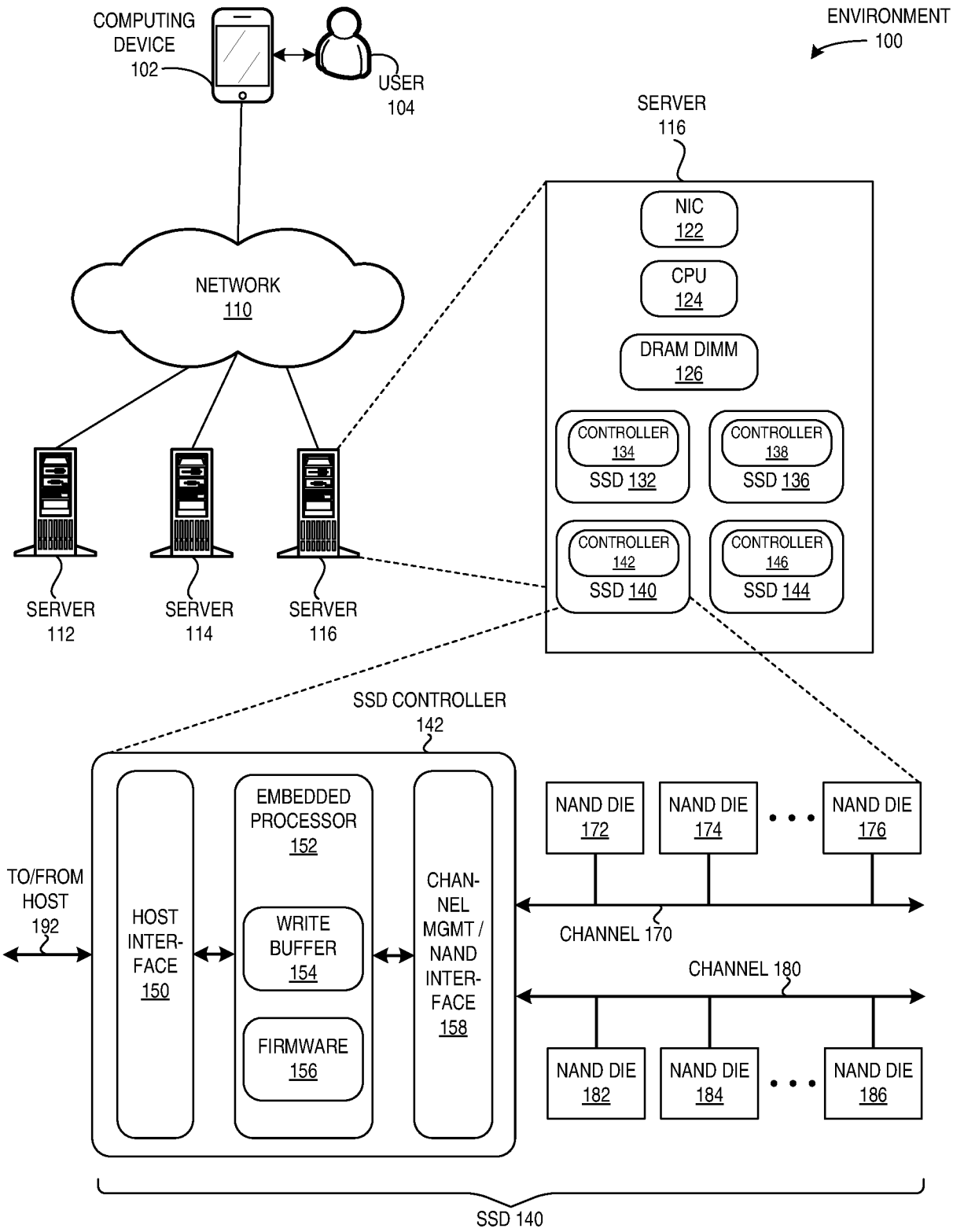

COMPUTING DEVICE 102
USER 104
ENVIRONMENT 100
SERVER 116
NIC 122
CPU 124
DRAM DIMM 126
NETWORK 110
CONTROLLER 134
SSD 132
CONTROLLER 138
SSD 136
CONTROLLER 142
SSD 140
CONTROLLER 146
SSD 144
SERVER 112
SERVER 114
SERVER 116
SSD CONTROLLER 142
EMBEDDED PROCESSOR 152
WRITE BUFFER 154
FIRMWARE 156
HOST INTER-FACE 150
CHAN-NEL MGMT / NAND INTER-FACE 158
TO/FROM HOST 192
NAND DIE 172
NAND DIE 174
NAND DIE 176
CHANNEL 170
CHANNEL 180
NAND DIE 182
NAND DIE 184
NAND DIE 186
SSD 140

FIG. 1

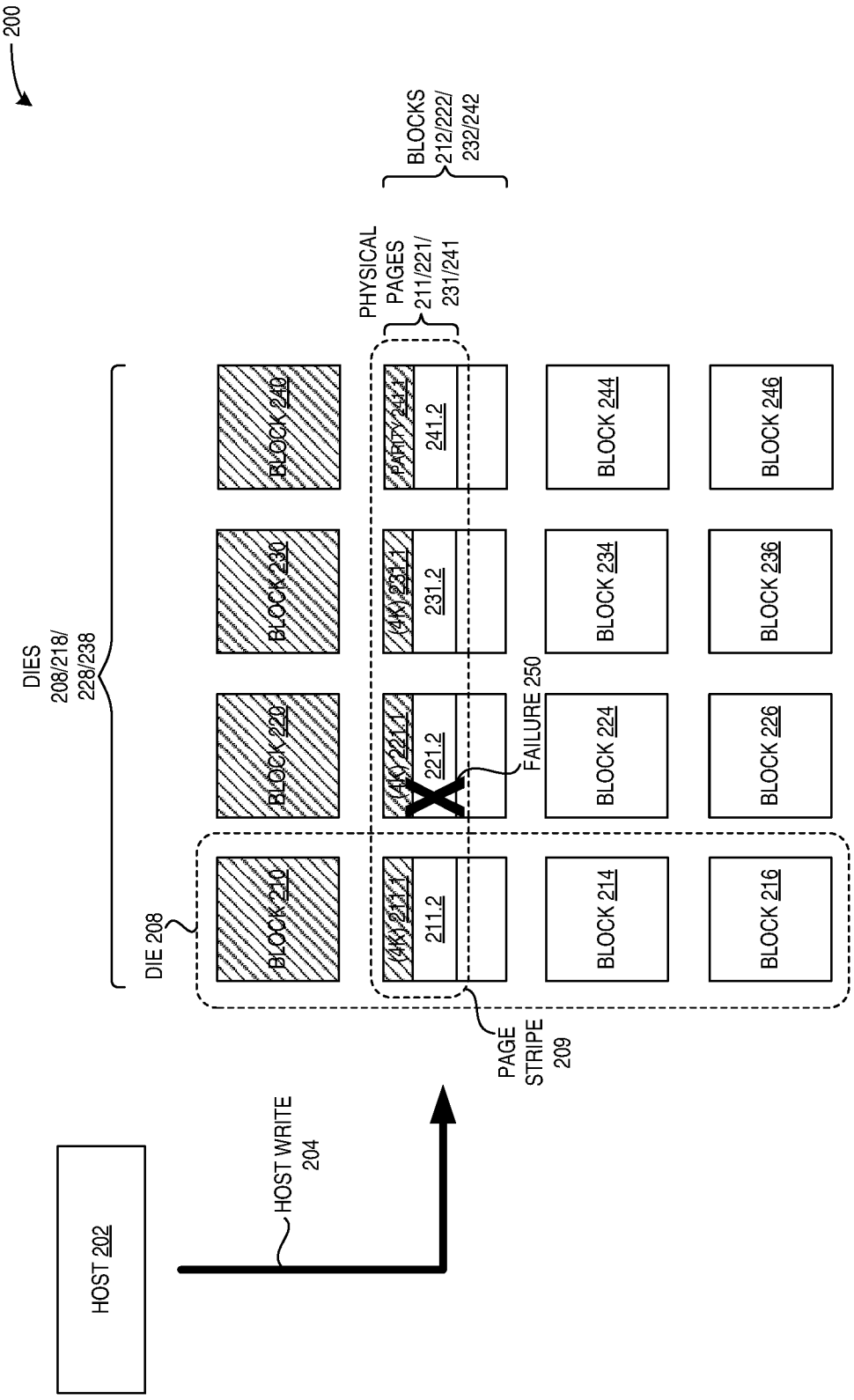


FIG. 2
(PRIOR ART)

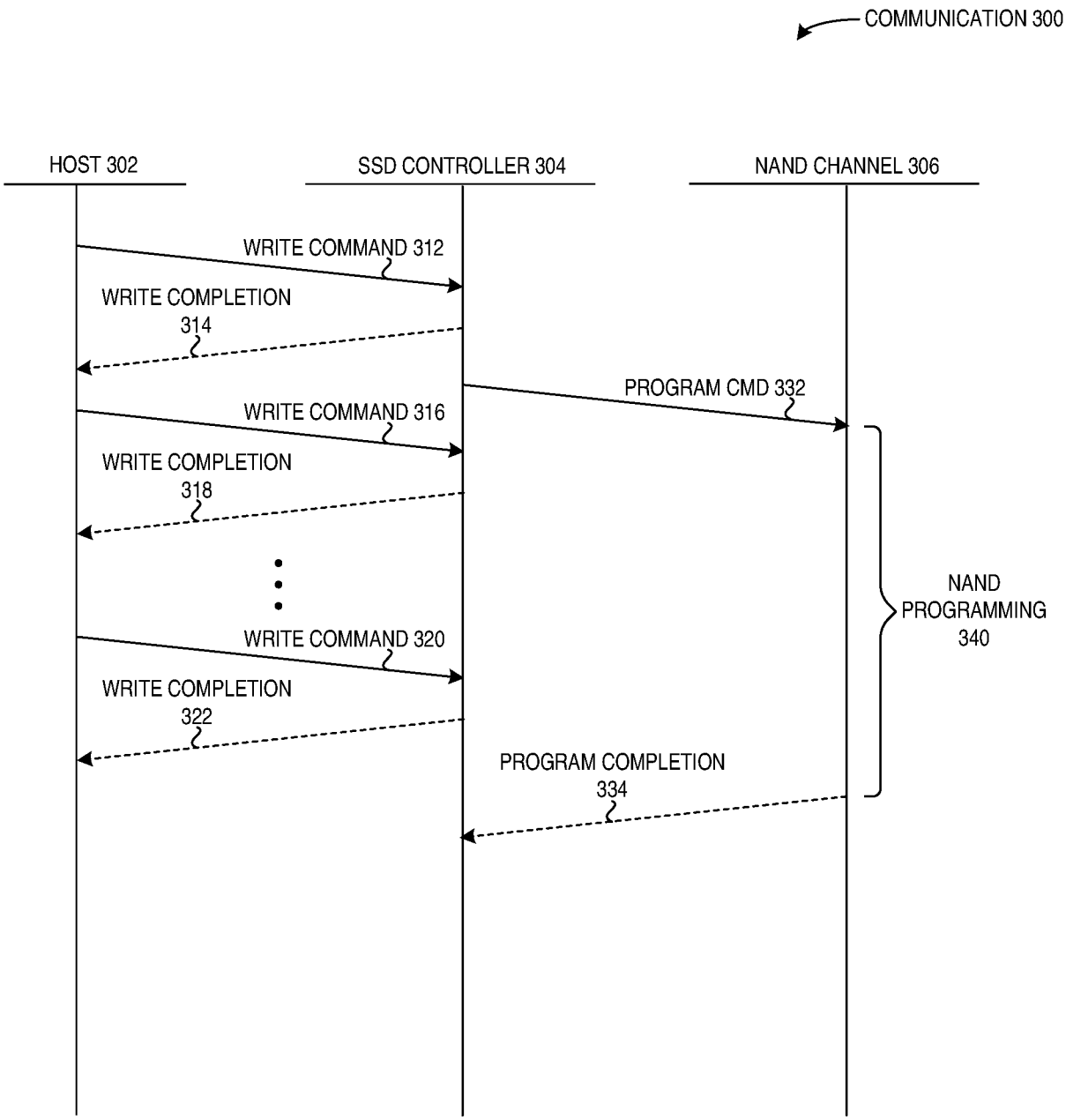


FIG. 3A
(PRIOR ART)

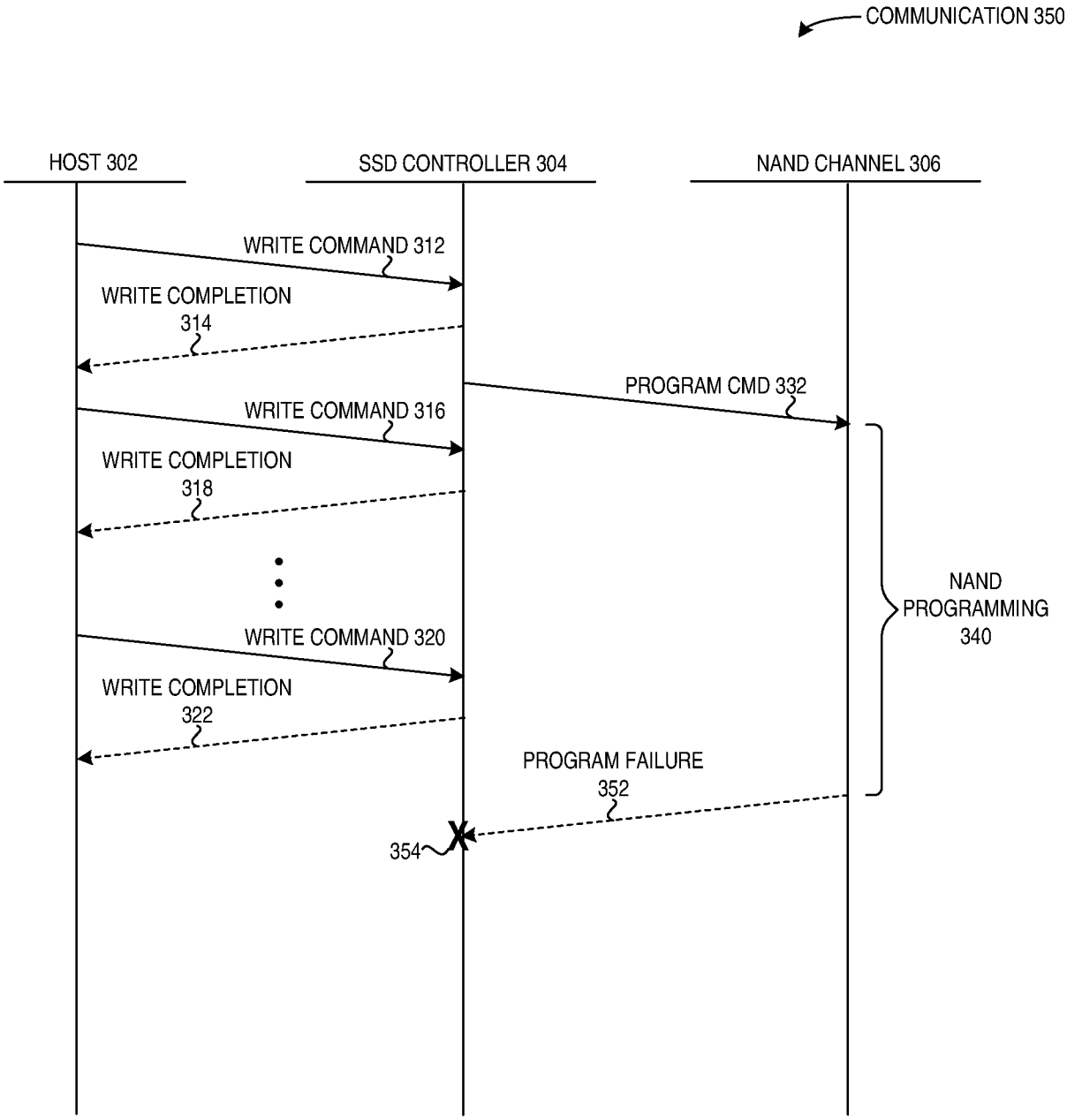


FIG. 3B
(PRIOR ART)

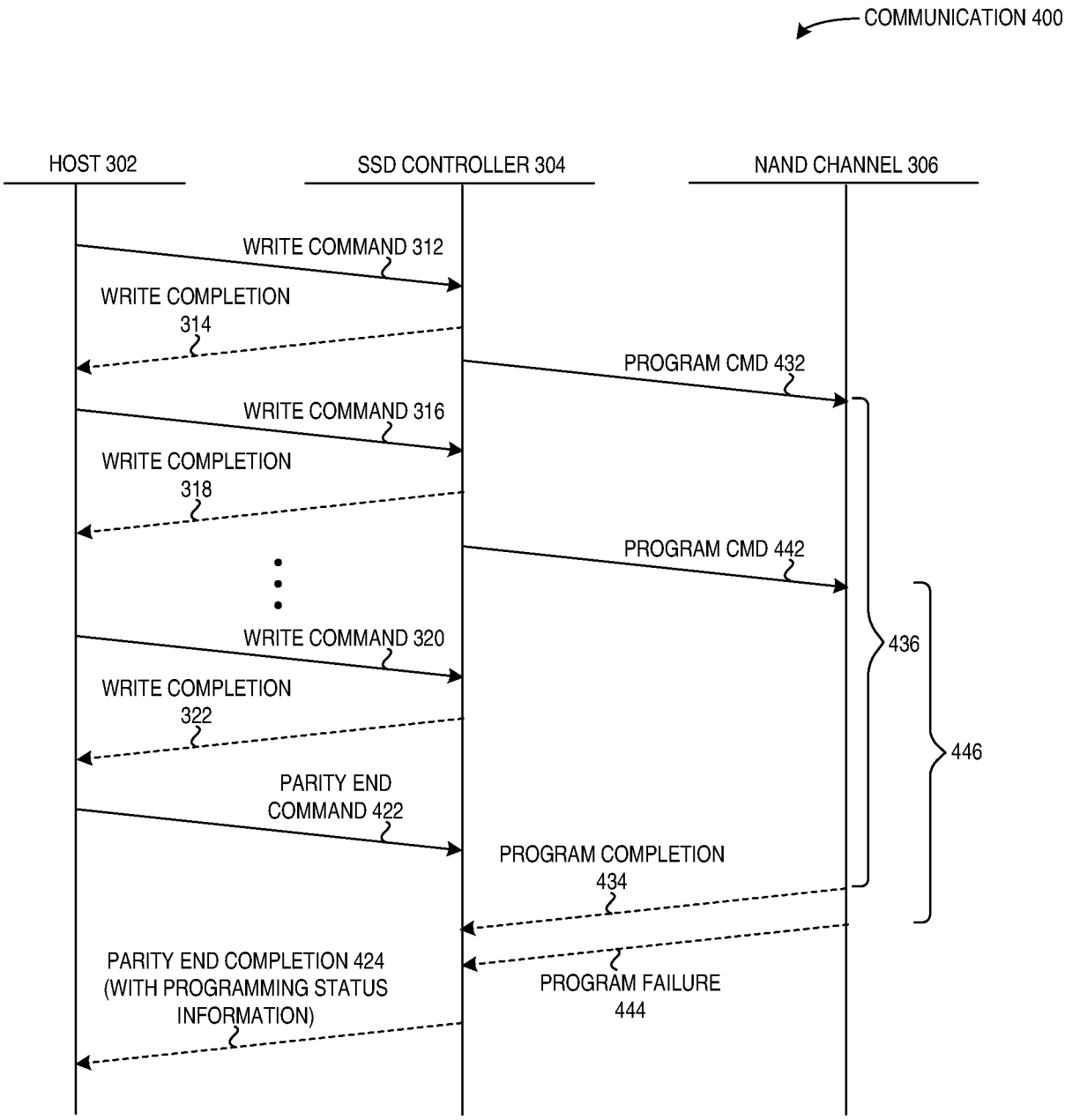

COMMUNICATION 400
HOST 302
SSD CONTROLLER 304
NAND CHANNEL 306
WRITE COMMAND 312
WRITE COMPLETION 314
PROGRAM CMD 432
WRITE COMMAND 316
WRITE COMPLETION 318
PROGRAM CMD 442
WRITE COMMAND 320
WRITE COMPLETION 322
PARITY END COMMAND 422
PROGRAM COMPLETION 434
PARITY END COMPLETION 424 (WITH PROGRAMMING STATUS INFORMATION)
PROGRAM FAILURE 444
436
446

FIG. 4

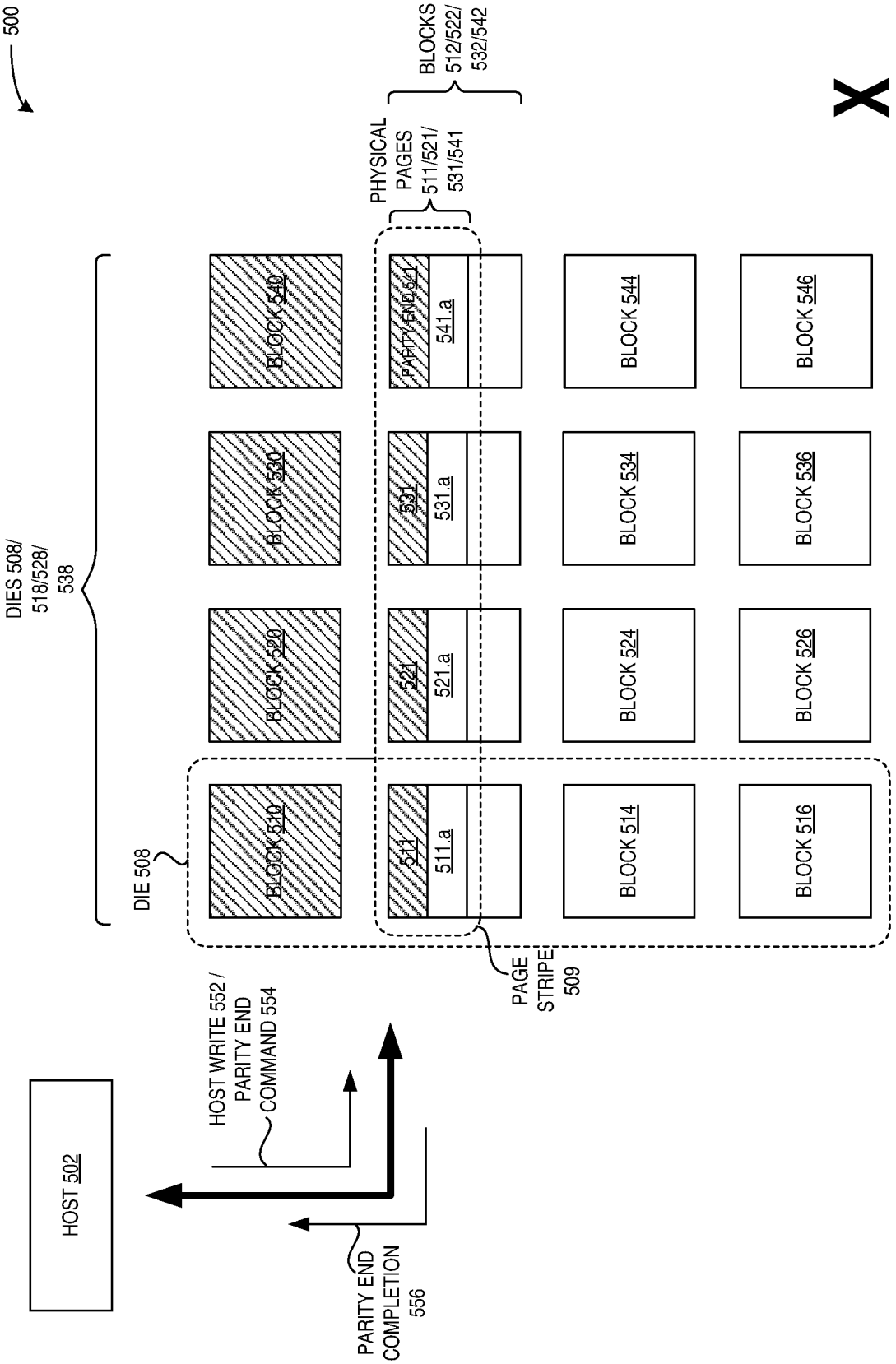

500
BLOCKS 512/522/ 532/542
PHYSICAL PAGES 511/521/ 531/541
DIES 508/ 518/528/ 538
DIE 508
BLOCK 510
BLOCK 520
BLOCK 530
BLOCK 540
511
511.a
521
521.a
531
531.a
PARITY END 541
541.a
BLOCK 514
BLOCK 524
BLOCK 534
BLOCK 544
BLOCK 516
BLOCK 526
BLOCK 536
BLOCK 546
PAGE STRIPE 509
HOST 502
HOST WRITE 552 / PARITY END COMMAND 554
PARITY END COMPLETION 556
X

FIG. 5

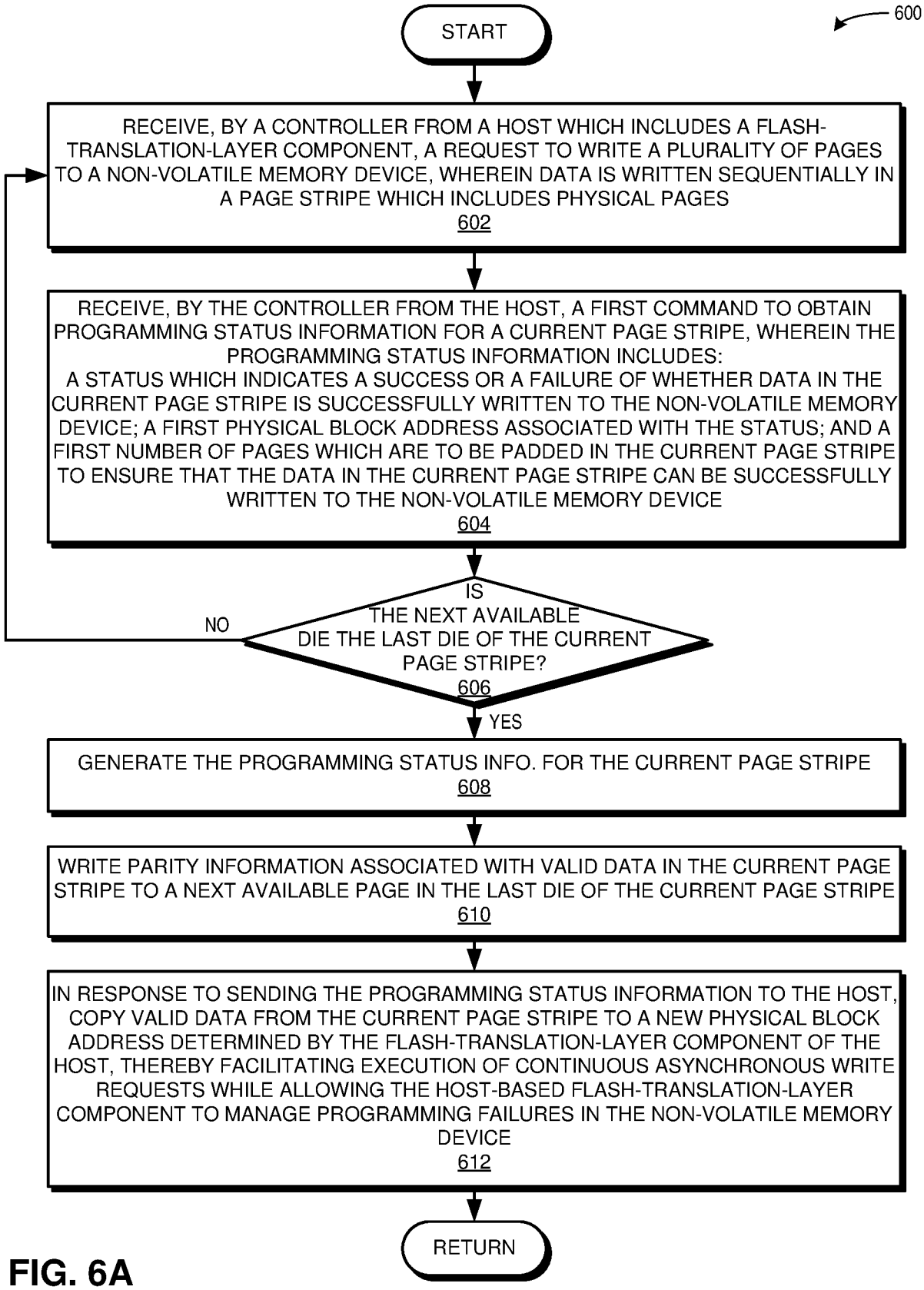

600
START
RECEIVE, BY A CONTROLLER FROM A HOST WHICH INCLUDES A FLASH-TRANSLATION-LAYER COMPONENT, A REQUEST TO WRITE A PLURALITY OF PAGES TO A NON-VOLATILE MEMORY DEVICE, WHEREIN DATA IS WRITTEN SEQUENTIALLY IN A PAGE STRIPE WHICH INCLUDES PHYSICAL PAGES
602
RECEIVE, BY THE CONTROLLER FROM THE HOST, A FIRST COMMAND TO OBTAIN PROGRAMMING STATUS INFORMATION FOR A CURRENT PAGE STRIPE, WHEREIN THE PROGRAMMING STATUS INFORMATION INCLUDES:
A STATUS WHICH INDICATES A SUCCESS OR A FAILURE OF WHETHER DATA IN THE CURRENT PAGE STRIPE IS SUCCESSFULLY WRITTEN TO THE NON-VOLATILE MEMORY DEVICE; A FIRST PHYSICAL BLOCK ADDRESS ASSOCIATED WITH THE STATUS; AND A FIRST NUMBER OF PAGES WHICH ARE TO BE PADDED IN THE CURRENT PAGE STRIPE TO ENSURE THAT THE DATA IN THE CURRENT PAGE STRIPE CAN BE SUCCESSFULLY WRITTEN TO THE NON-VOLATILE MEMORY DEVICE
604
IS THE NEXT AVAILABLE DIE THE LAST DIE OF THE CURRENT PAGE STRIPE?
606
NO
YES
GENERATE THE PROGRAMMING STATUS INFO. FOR THE CURRENT PAGE STRIPE
608
WRITE PARITY INFORMATION ASSOCIATED WITH VALID DATA IN THE CURRENT PAGE STRIPE TO A NEXT AVAILABLE PAGE IN THE LAST DIE OF THE CURRENT PAGE STRIPE
610
IN RESPONSE TO SENDING THE PROGRAMMING STATUS INFORMATION TO THE HOST, COPY VALID DATA FROM THE CURRENT PAGE STRIPE TO A NEW PHYSICAL BLOCK ADDRESS DETERMINED BY THE FLASH-TRANSLATION-LAYER COMPONENT OF THE HOST, THEREBY FACILITATING EXECUTION OF CONTINUOUS ASYNCHRONOUS WRITE REQUESTS WHILE ALLOWING THE HOST-BASED FLASH-TRANSLATION-LAYER COMPONENT TO MANAGE PROGRAMMING FAILURES IN THE NON-VOLATILE MEMORY DEVICE
612
RETURN

FIG. 6A

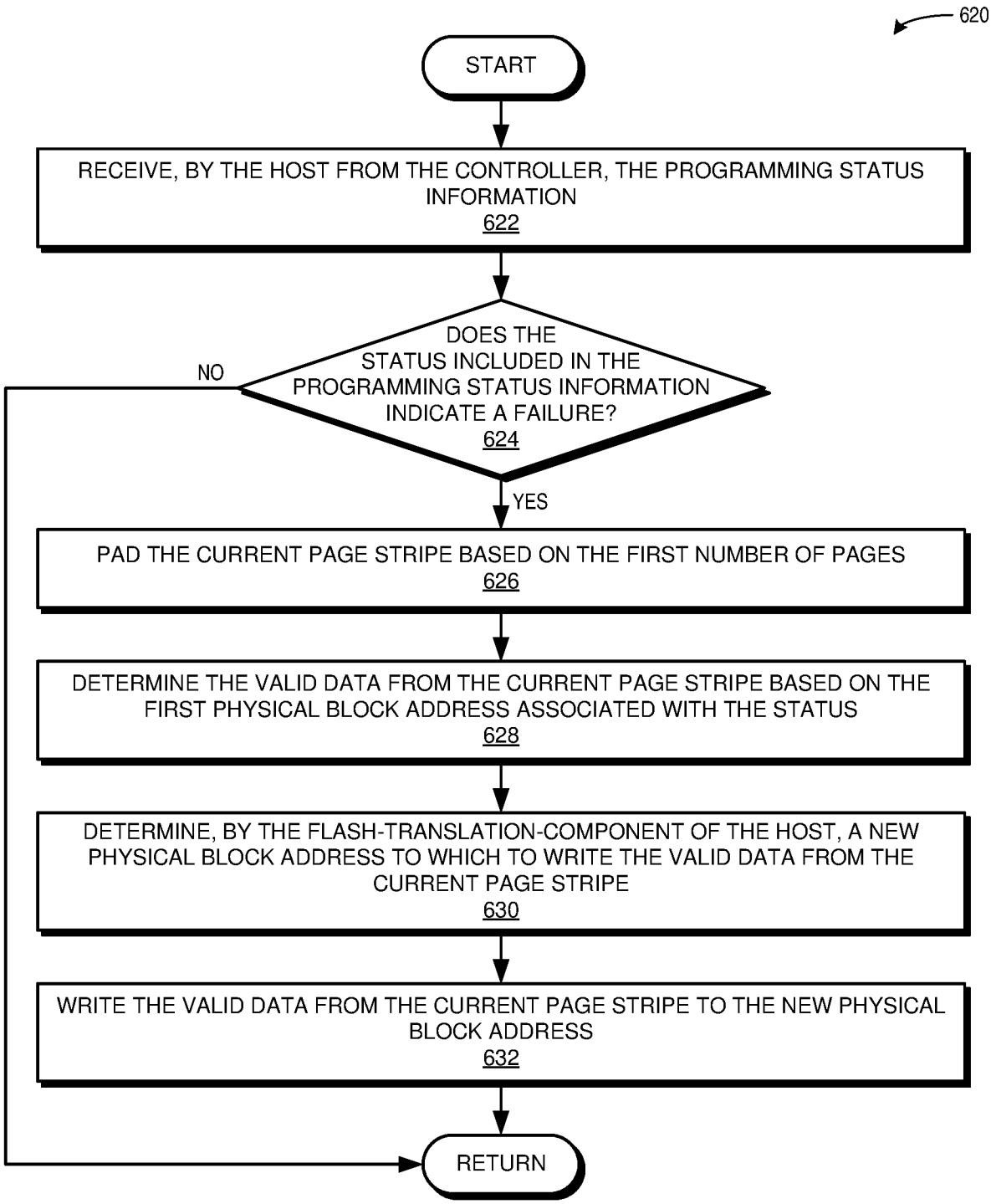

620
START
RECEIVE, BY THE HOST FROM THE CONTROLLER, THE PROGRAMMING STATUS INFORMATION
622
DOES THE STATUS INCLUDED IN THE PROGRAMMING STATUS INFORMATION INDICATE A FAILURE?
624
NO
YES
PAD THE CURRENT PAGE STRIPE BASED ON THE FIRST NUMBER OF PAGES
626
DETERMINE THE VALID DATA FROM THE CURRENT PAGE STRIPE BASED ON THE FIRST PHYSICAL BLOCK ADDRESS ASSOCIATED WITH THE STATUS
628
DETERMINE, BY THE FLASH-TRANSLATION-COMPONENT OF THE HOST, A NEW PHYSICAL BLOCK ADDRESS TO WHICH TO WRITE THE VALID DATA FROM THE CURRENT PAGE STRIPE
630
WRITE THE VALID DATA FROM THE CURRENT PAGE STRIPE TO THE NEW PHYSICAL BLOCK ADDRESS
632
RETURN

FIG. 6B

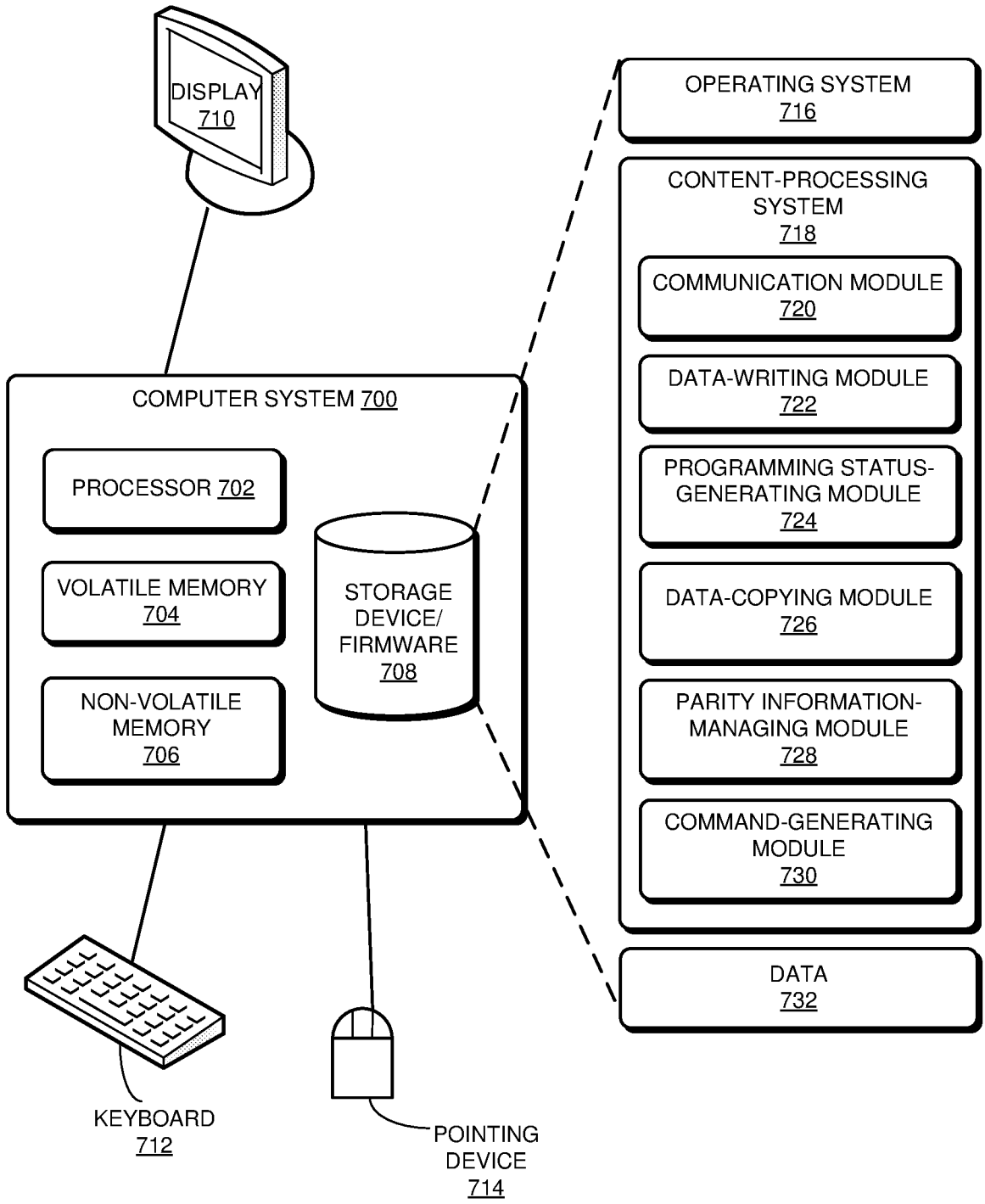

DISPLAY
710
COMPUTER SYSTEM 700
PROCESSOR 702
VOLATILE MEMORY
704
NON-VOLATILE
MEMORY
706
STORAGE
DEVICE/
FIRMWARE
708
KEYBOARD
712
POINTING
DEVICE
714
OPERATING SYSTEM
716
CONTENT-PROCESSING
SYSTEM
718
COMMUNICATION MODULE
720
DATA-WRITING MODULE
722
PROGRAMMING STATUS-
GENERATING MODULE
724
DATA-COPYING MODULE
726
PARITY INFORMATION-
MANAGING MODULE
728
COMMAND-GENERATING
MODULE
730
DATA
732

FIG. 7

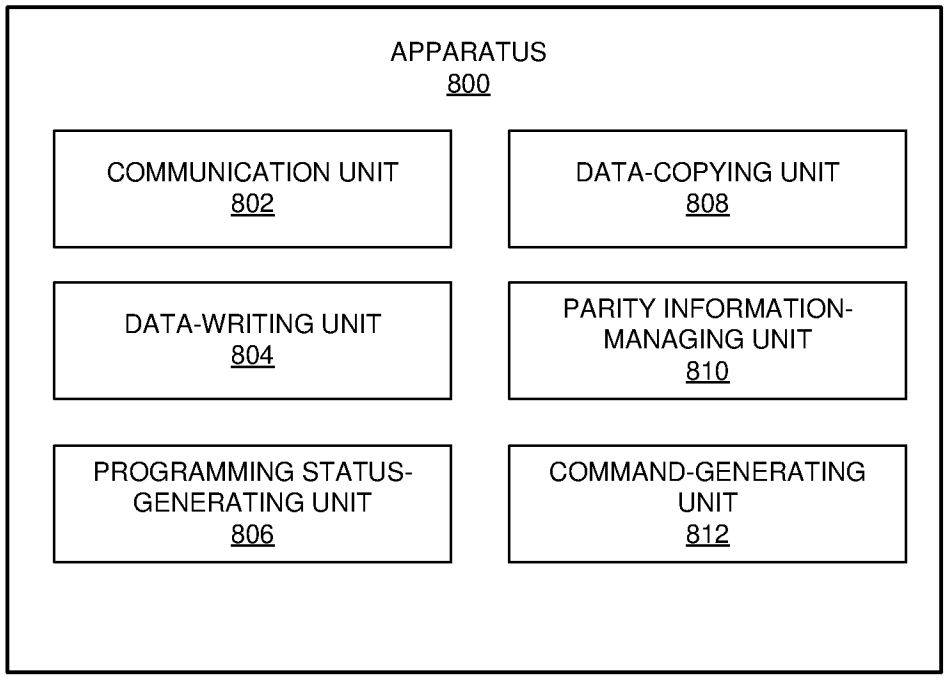

APPARATUS
800
COMMUNICATION UNIT
802
DATA-COPYING UNIT
808
DATA-WRITING UNIT
804
PARITY INFORMATION-MANAGING UNIT
810
PROGRAMMING STATUS-GENERATING UNIT
806
COMMAND-GENERATING UNIT
812

FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/CN2018/112638

A. CLASSIFICATION OF SUBJECT MATTER

G06F 3/06(2006.01)i; G06F 12/02(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, G11C

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

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

CNKI,CNPAT,WPI,EPODOC,IEEE: Flash translation layer, open channel, SSD, programme, success, failure, page, die, host, valid, write

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	CN 101800071 A (SAMSUNG ELECTRONICS CO., LTD.) 11 August 2010 (2010-08-11) description, paragraphs [0049], [0069]-[0079] and figures 1-6	1-20
A	US 2016224089 A1 (SILICON MOTION INC.) 04 August 2016 (2016-08-04) the whole document	1-20
A	CN 108255426 A (BEIJING LENOVO CO., LTD.) 06 July 2018 (2018-07-06) the whole document	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

"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
09 July 2019	**25 July 2019**

Name and mailing address of the ISA/CN	Authorized officer
National Intellectual Property Administration, PRC **6, Xitucheng Rd., Jimen Bridge, Haidian District, Beijing 100088** **China**	**SHU,Qi**
Facsimile No. **(86-10)62019451**	Telephone No. **86-(10)-53961220**

INTERNATIONAL SEARCH REPORT
Information on patent family members

International application No.
PCT/CN2018/112638

Patent document cited in search report	Publication date (day/month/year)	Patent family member(s)	Publication date (day/month/year)
CN 101800071 A	11 August 2010	KR 20100091379 A	19 August 2010
		CN 101800071 B	18 March 2015
		US 2010205517 A1	12 August 2010
US 2016224089 A1	04 August 2016	TW I569144 B	01 February 2017
		US 9563249 B2	07 February 2017
		TW 201629781 A	16 August 2016
		CN 106155874 A	23 November 2016
		CN 106155874 B	21 December 2018
CN 108255426 A	06 July 2018	None	