Technical Disclosure Commons

Defensive Publications Series

July 2020

High-Performance Low-Overhead Stochastic Wear Leveling of Flash Memory By Comparing Age of a Block with Age of a Randomly Selected Block

Chris Phoenix

High-Performance Low-Overhead Stochastic Wear Leveling of Flash Memory By Comparing Age of a Block with Age of a Randomly Selected Block

Abstract:

This publication describes systems and techniques to implement a stochastic-based wear leveling scheme for block erasures to extend the usable life of flash memory and other types of electrically erasable programmable read-only memory (EEPROM). Some existing approaches for wear leveling are resource intensive and may be merged with a file system that also attempts to distribute write operations, which impacts both the write distribution and the erasure wear leveling. In contrast, described systems and techniques can be implemented separately from file system routines, including at the hardware level. Initially, a block of a memory is identified for erasure. The erasure age of the identified block is compared to a first threshold, which is based on an average age of the blocks across the memory. A targeted block is determined using a random process. The erasure age of the targeted block is compared to a second threshold, which is based on the erasure age of the identified block. If both comparisons are positive, the data of the targeted block is moved to the identified block to promote wear leveling, the targeted block is erased, and a mapping table is updated to reflect the block-address swap. If not, block-address swapping is skipped for this erasure operation. In these manners, the described wear leveling scheme can be implemented efficiently and can approach optimum wear leveling performance.

Keywords:

electrically erasable programmable read-only memory (EEPROM), flash memory, solid-state drive (SSD), wear leveling, erase cycles, erasures, blocks, mapping, physical, virtual, age, experience count, hot block, old block, young block, cold block, swap, exchange

Background:

Many electronic devices, such as notebook computers and smartphones, include flash memory for nonvolatile storage instead of spinning magnetic disks (e.g., a "hard drive"). Typically, flash memory is smaller, uses less power, and is more reliable than a hard drive. A solid-state drive (SSD) made from flash memory, however, can have a shorter usable lifespan than a hard drive. Flash memory disks are organized into blocks that are independently erased. Each block is associated with a maximum number of erase cycles before the block reaches a level of unreliability that is deemed nonfunctional. Thus, the usable lifetime of a flash memory device can be increased by erasing each block an approximately equal number of times. Algorithms to evenly distribute erasure operations across blocks of the flash memory are referred to as "wear leveling."

Some existing implementations of wear leveling algorithms are complex and difficult to use. For instance, wear leveling algorithms are separated into various categories, including static wear leveling and dynamic wear leveling. Existing algorithms can be resource intensive as they approach an optimal wear leveling pattern (e.g., as their effectiveness approaches a substantially equal number of erasures per block). Further, some approaches integrate wear leveling with the management of the file system (e.g., "filesystem management"), which may have access to greater processing capabilities to handle the complexity of existing wear leveling algorithms. Using a complex wear leveling algorithm can lead to more complications, increased costs, and even lower

performance as compared to the schemes described in this paper. For example, filesystem management is responsible for distributing write operations across a memory. When the computations for write-operation distribution are merged with those for erasure wear leveling, the efficiency of both the write distribution and the wear leveling can be decreased.

It is desirable to provide systems and techniques that enable low-resource aspects of existing wear leveling algorithms to be used with simpler computational strategies while avoiding reliance on high-resource data structures and computations. Two simple data structures typically found in existing algorithms are sufficient to implement the present stochastic algorithm. Existing algorithms typically store, in association with each block of the flash memory, a count of the number of times the associated block has been erased. These algorithms also store and manage a table that maps each "virtual" block number (e.g., address or index) to a distinct "physical" block number. This mapping enables data to be moved between blocks in physical storage without changing the virtual address of the data. The schemes described in this paper can leverage these two relatively low-resource data structures in conjunction with a simple stochastic computational scheme.

Description:

A stochastic-based wear leveling scheme for block erasures is described that extends the usable life of flash memory and other types of electrically erasable programmable read-only memory (EEPROM). During processing operations, a block of memory that has been filled with unneeded data is identified for erasure. The erasure age of the identified block is compared to a first threshold, which is based on an average age of the blocks across the memory. A block that is targeted for potential data movement and address swapping is determined using a random process.

The erasure age of the targeted block is compared to a second threshold, which is based on the erasure age of the identified block. Each respective threshold can further be based on a respective tunable hysteresis parameter.

If both comparison operations are affirmative, the physical address of the identified block is swapped with the physical address of the targeted block to facilitate wear leveling. To do so, the identified block is erased, the data of the targeted block is moved to the identified block, and the targeted block is erased. The virtual address mapping is also updated. If either comparison is negative, on the other hand, the addresses are not swapped for this erasure operation and only the identified block is erased. As described herein, the scheme can use a table that tracks the numbers of erasures per block and another table that maps virtual addresses to physical ones. Given a reasonably high number of erase operations per block, wear leveling performance can approach that of an ideal algorithm over time. In these manners, the wear leveling scheme can be implemented efficiently and can provide a substantially equalized erasure level across the memory.

An example environment having a flash memory system in which wear leveling can be implemented is described with reference to Fig. 1. An example wear leveling scheme with first and second thresholds is described with reference to a schematic diagram of Fig. 2. An example wear leveling technique is described with reference to a flow chart of Fig. 3. A Python code example for an approach to the described wear leveling principles is set forth thereafter. This paper concludes with some example statistics indicating the viability of the described systems and techniques.

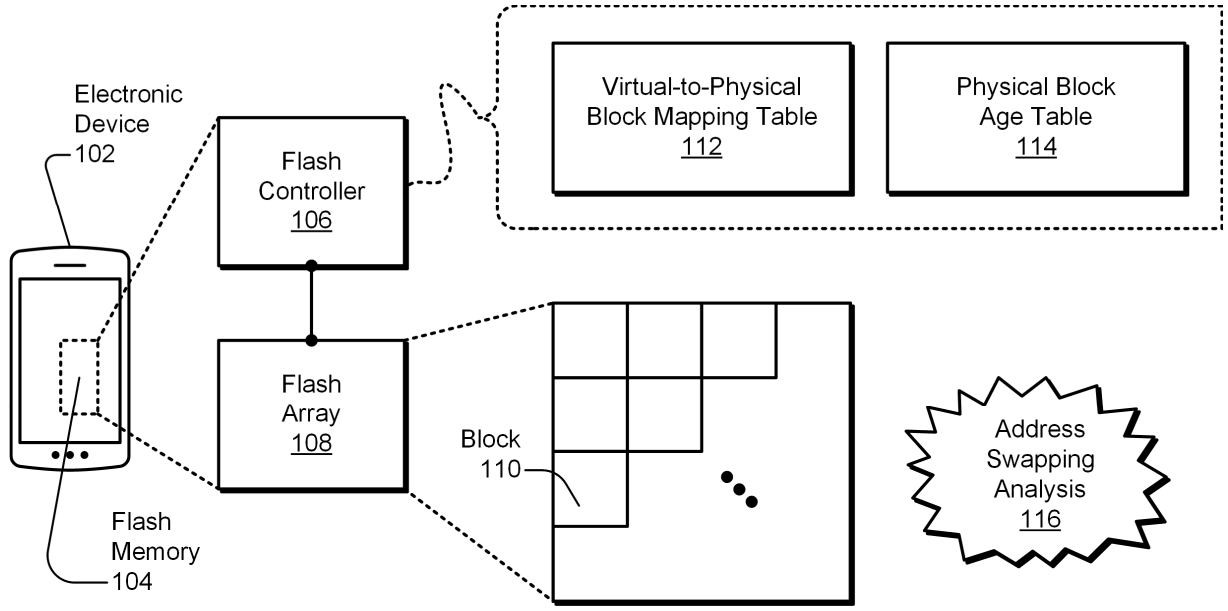

Fig. 1. Electronic device with an example wear leveling flash memory system.

As shown in Fig. 1, an electronic device 102 includes at least one flash memory 104. Although depicted as a smartphone, the electronic device 102 can be realized as any computing device. Examples include a notebook computer, a wearable device like a smartwatch, a desktop computer, a server computer or portion thereof (including server racks in data centers), devices embedded in machines such as vehicles, Internet of Things (IoTs) devices, medical devices, and so forth. Although principles are described in terms of a flash memory 104 (e.g., NAND or NOR flash), the principles are applicable to electrically erasable programmable read-only memory (EEPROM) generally and to SSDs, USB drives, and so forth. Further, the described systems and techniques can be applied to other types of memory, like phase-change memory or any memory in which wear leveling is applicable, such as to extend the life of a memory device or for another purpose.

As illustrated in Fig. 1, the flash memory 104 includes a flash controller 106 and a flash array 108. The data storage units of the flash array 108 are separated into blocks 110 or other portions of the flash array 108. The flash controller 106 includes or has access to a virtual-to-

physical block mapping table 112 and a physical block age table 114. The flash controller 106 can be implemented in hardware, firmware, software, or some combination thereof. Thus, although the flash controller 106 is shown coupled to the flash array 108 as part of the flash memory 104, the flash controller 106 can be implemented in a separate module or other portion of the electronic device 102, including as part of a simplified file management system or an operating system (OS).

A filesystem management component or an OS component can access the flash memory 104 using a virtual address (i.e., a number that is indirectly indicative of a block index or block location) for a block 110. The flash controller 106 can convert the virtual address to a physical address (e.g., a number that may be "directly" indicative of a block index or block location) of the block 110 using the virtual-to-physical block mapping table 112. The physical block age table 114 includes an entry for each block 110 in which each entry stores an age of the associated block. The age can be provided in terms of a number of erase operations that have been applied to the associated block. A total number of erasures can be determined from a sum of the block entries or accumulated as a separate entry. The virtual-to-physical block mapping table 112 and the physical block age table 114 can be realized and maintained using an existing approach.

In example operations, the flash controller 106 is instructed to erase a block 110 (identified by its virtual address) by higher-level algorithms that are outside the scope of this paper. The flash controller 106 performs an address swapping analysis 116 to determine if the physical address of the identified block is to be swapped with the physical address of a targeted block to facilitate wear leveling. An example address swapping analysis 116 is depicted schematically in Fig. 2.

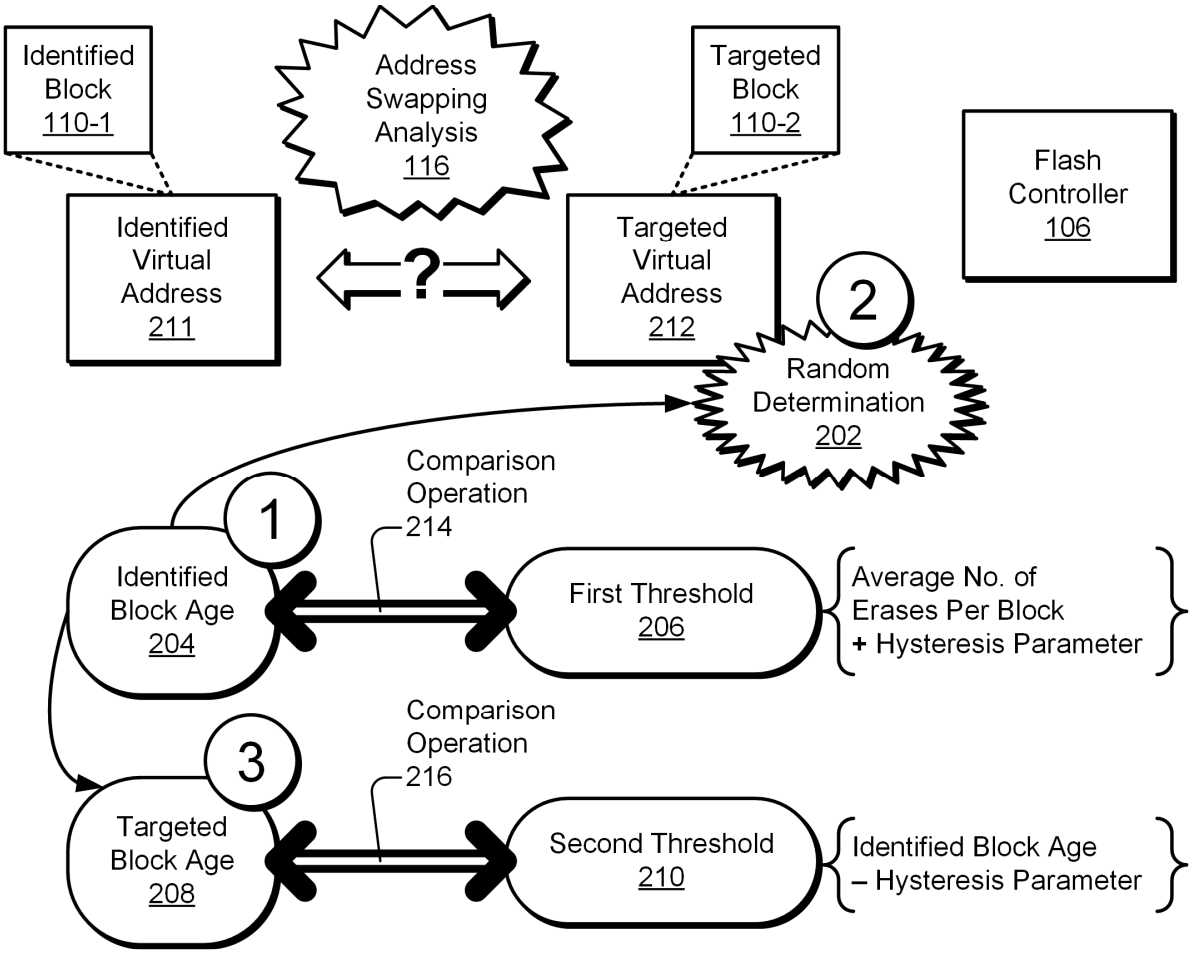

Fig. 2. Schematic diagram of an example address swapping analysis for wear leveling.

Fig. 2 depicts an example address swapping analysis 116 between an identified virtual address 211 and a targeted virtual address 212. Here, an identified block 110-1, which is mapped from the identified virtual address 211, is scheduled for erasing. The analysis entails determining whether the virtual-to-physical mapping between the identified block 110-1 and a targeted block 110-2 is going to be swapped, that is, whether the identified virtual address 211 is to be swapped with the targeted virtual address 212. If so, this address swapping involves moving data from the targeted block 110-2 corresponding to the targeted virtual address 212 to the identified block 110-1 corresponding to the identified virtual address 211 to promote wear leveling in conjunction with,

or as part of, the erasing operation. The address swapping analysis 116 can be described in terms of three parts.

In a first of the three parts (as indicated with an encircled numeral "1"), a first comparison operation 214 is performed. The flash controller 106 compares an age 204 of the identified block 110-1 to a first threshold 206. The first threshold 206 can be computed based on a combination of the average number of erases per block across the flash array 108 (of Fig. 1) and a first hysteresis parameter (e.g., calculated through an arithmetic addition). An example of the first hysteresis parameter is described below with reference to the Python code using a variable "ABOVE." This comparison operation involves comparing the age of the identified block 110-1 to what its age would be if it were "ideally" wear leveled along with the other blocks of the flash array 108. The first comparison operation 214 therefore determines if the identified block 110-1 is "too old," or has been erased more than the average number of times per block. Each time an identified block 110-1 is to be erased, if the identified block 110-1 is not "too old," then the identified block 110-1 can be erased without proceeding further in the address swapping analysis 116. Avoiding performance of the second and third parts of the analysis can reduce utilization of processing resources. On the other hand, if the identified block 110-1 is "too old," then the address swapping analysis 116 includes randomly choosing a virtual address corresponding to a targeted block 110-2.

Accordingly, in a second of three parts (which is indicated with an encircled numeral "2"), the flash controller 106 performs a random determination 202 to randomly select a targeted virtual address 212 that corresponds to a targeted block 110-2 of the flash array 108. Different stochastic techniques can be implemented to realize the random determination 202. The random determination 202 can be implemented in software, firmware, hardware, or some combination thereof. For example, a hardware implementation may utilize a hash of a fast hardware counter.

Various numbers drawn from device commands may be used as seeds for a pseudorandom generator; however, basing the pseudorandom numbers entirely on externally supplied commands may leave the device vulnerable to unlucky or hostile command sequences.

In a third of the three parts (as indicated with an encircled numeral "3"), a second comparison operation 216 is performed to determine if the targeted block 110-2 is "young enough." The flash controller 106 compares an age 208 of the targeted block 110-2 to a second threshold 210. The second threshold 210 can be computed based on a combination of the identified block age 204 and a second hysteresis parameter (e.g., calculated using an arithmetic subtraction). An example of the second hysteresis parameter is described below with reference to the Python code using a variable "BELOW." In operation, if the targeted block 110-2 is "young enough," then the flash controller 106 swaps the virtual-to-physical address mapping of the two blocks and performs a data move to preserve the contents of the targeted block 110-2. The flash controller 106 updates the virtual-to-physical block mapping table 112 to swap the two addresses.

Thus, the factors "too old" from the first part and "young enough" from the third part can be computed based on simple arithmetic using two comparison operations. These two comparisons are described above respectively in terms of first and second thresholds, each of which can have at least one hysteresis offset. In alternative implementations, two or more parts of the analysis may be performed at least partially in parallel. For example, to accelerate the erasing procedure, the random determination 202 of the second part can be performed while the first operation 214 of the first part is being performed. Before the example Python code is set forth further below, an example wear leveling technique is described with reference to Fig. 3.

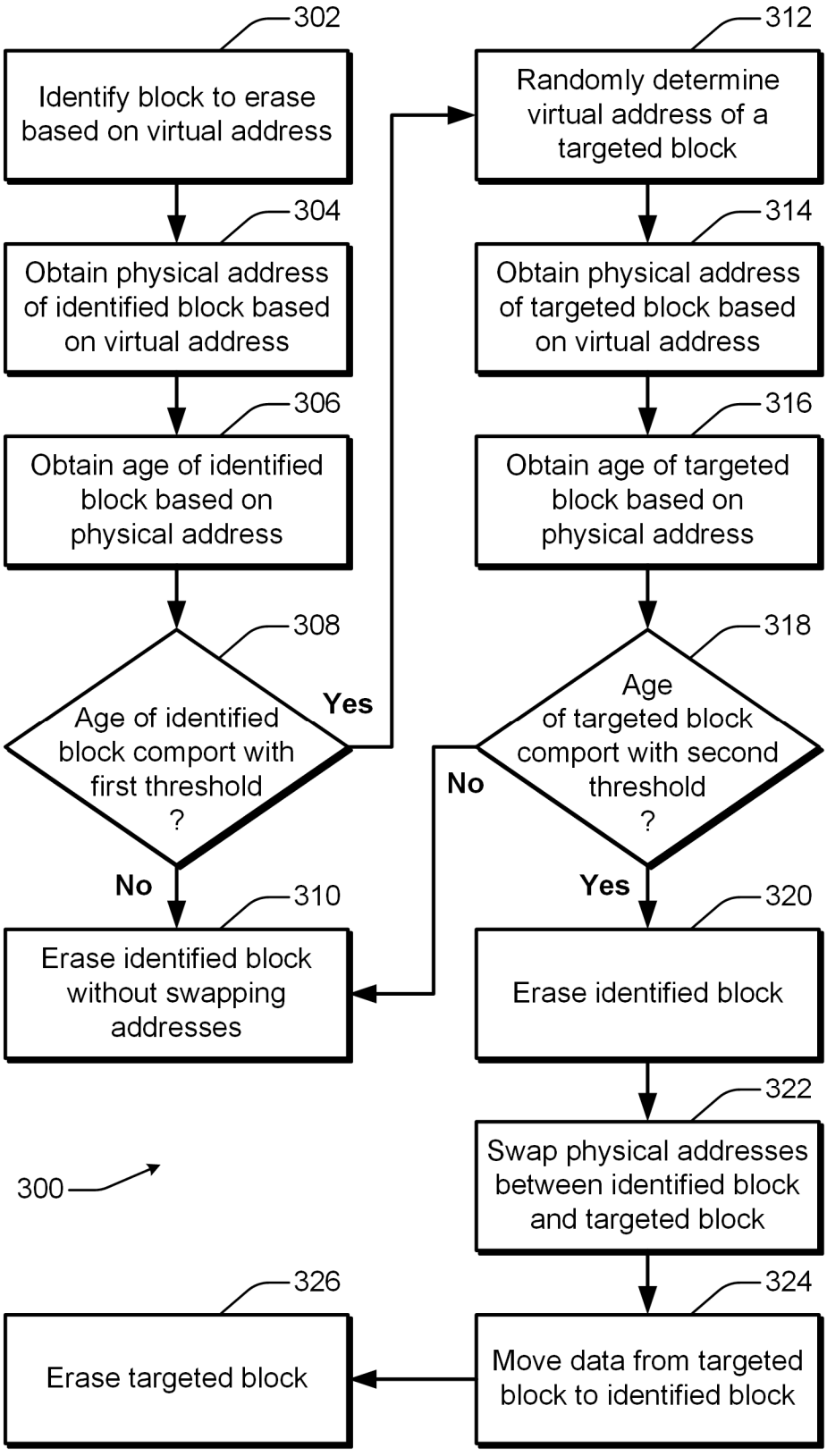

Fig. 3. Flowchart illustrating an example wear leveling process.

Fig. 3 depicts an example technique 300 to implement wear leveling using the systems and schemes that are described in this document. The technique 300 includes thirteen (13) operations 302-326 that can be performed by, for example, the flash controller 106. Although the operations are described in a particular order, they may be performed in different orders, or they may be performed partially or fully overlapping. Further, the technique 300 can include more or fewer operations than those that are described or shown.

At operation 302, a block 110-1 to erase is identified based on a virtual address 211. At operation 304, a physical address of the identified block 110-1 is obtained based on the virtual address 211 and using the virtual-to-physical block mapping table 112. At operation 306, an age 204 of the identified block 110-1 is obtained based on the physical address and using the physical block age table 114.

At operation 308, with the first comparison operation 214, the flash controller 106 determines if the age 204 of the identified block 110-1 comports with a first threshold 206. If the age 204 is less than the first threshold 206 (e.g., is not “too old”), the physical block 110-1 corresponding to the identified virtual address 211 is erased at operation 310 without swapping addresses, and the technique 300 can conclude for this erasure operation. If, on the other hand, the flash controller 106 determines at operation 308 that the age 204 of the identified block 110-1 is greater than the first threshold 206 (e.g., is “too old”), the technique 300 continues at operation 312.

At operation 312, the flash controller 106 randomly (including pseudo-randomly) determines a targeted virtual address 212 of a targeted block 110-2 using a stochastic mechanism, which can be based in hardware. At operation 314, a physical address of the targeted block 110-2 is obtained based on the targeted virtual address 212 thereof and using the virtual-to-physical

block mapping table 112. Alternatively, a physical address can be randomly generated at operation 312—e.g., if the virtual-to-physical block mapping table 112 is bidirectional. At operation 316, the flash controller 106 obtains an age 208 of the targeted block 110-2 based on the physical address thereof and using the physical block age table 114.

At operation 318, with the second comparison operation 216, the flash controller 106 determines if the age 208 of the targeted block 110-2 comports with a second threshold 210. If the age 208 of the targeted block 110-2 is greater than the second threshold 210 (e.g., is not "young enough"), the flash controller 106 erases the identified block 110-1 corresponding to the identified virtual address 211 at operation 310 without modifying the mapping to swap physical addresses, and the technique 300 can conclude for this erasure operation. Skipping address swapping if the randomly selected targeted virtual address 212 is not "young enough" can increase efficiency by not searching for a suitable block responsive to each erasure operation. Given the high number of operations across a multitude of blocks, wear leveling performance can nonetheless approach that of an ideal algorithm over time. If, on the other hand, the flash controller 106 determines at operation 318 that the age 208 of the targeted block 110-2 is less than the second threshold 210 (e.g., is "young enough"), the physical address mapping is swapped as part of an erasure operation starting at operation 320.

At operation 320, the identified block 110-1 is erased. At operation 322, the flash controller 106 swaps the physical addresses between the identified block 110-1 and the targeted block 110-2. To do so, the flash controller 106 can swap the physical addresses mapped to the identified virtual address 211 and the targeted virtual address 212 in the virtual-to-physical block mapping table 112. At operation 324, data is moved from the targeted block 110-2 to the identified block 110-1 to promote wear leveling. Instead of the illustrated order, the data movement of operation 324 can

precede the map table updating of operation 322 to maintain a pointer to the targeted data throughout the process, but this results in the moved data being temporarily under the identified virtual address. The targeted block 110-2 can then be erased at operation 326. In an alternative implementation to those depicted in the operations 320, 324, and 326, the contents of the targeted block 110-2 can be moved to temporary (e.g., RAM) storage, both identified and targeted blocks 110-1 and 110-2 can be erased simultaneously, and the contents of the temporary storage can then be moved to the identified block 110-1. In either of these example operational orders, the targeted block 110-2 can be left empty and available for new storage.

With the description of Fig. 3 concluded, an example approach to wear leveling using two comparison operations, two respective thresholds, and a randomly determined targeted block is set forth below using Python code. However, described approaches can be implemented in other languages and/or in hardware. For this Python code, assume the following variable meanings:

virt: the virtual address to be erased. This corresponds to the identified virtual address 211 and the identified physical block 110-1 in the description above.

age[]: The number of erases of each block, which is initially 0 for each.

map[]: The number at each position is the current physical address of the data at that virtual address.

erases: The total number of block erases, which is equal to the sum of age[].

data[]: The actual data contents of the block to be erased. Moving data when swapping addresses can entail erasing the identified block 110-1, copying data from the targeted block 110-2 to the identified block 110-1, and erasing the targeted block 110-2. Or alternatively,

moving data can entail reading data from the targeted block 110-2 into RAM, erasing identified and targeted blocks 110-1 and 110-2 together or simultaneously (which may save time), and writing data from the RAM into the just-erased identified block 110-1.

BLOCKS: the total number of blocks in the memory being wear leveled.

ABOVE, BELOW: Tuned hysteresis parameters for the first and second thresholds, respectively. In some cases, each parameter can be set to approximately the square root of the maximum erase cycles per block, with ABOVE several times larger than BELOW.

For each block erasure operation, the following Python code is executed. With this code, no data structures other than age[] and map[] are needed, and there are no loops.

```
def erase(virt):
    physically_clear(virt)
    erases += 1
    phys = map[virt]
    age[phys] += 1
    if age[phys] <= erases/BLOCKS + ABOVE:
        return
    target_virt = randint(0, BLOCKS-1)
    target_phys = map[target_virt]
    if age[target_phys] >= age[phys] - BELOW:
        return
    data[phys] = data[target_phys]
```

```
map[target_virt], map[virt] = map[virt], map[target_virt]
physically_clear(target_virt)
```

Example implementations of the described systems and techniques have been modeled and analyzed. With an erase endurance of 100,000 cycles, the described algorithms are sufficient for wear leveling that can achieve >99% of the maximum number of erases that would be possible with ideal, cost-free wear leveling. Further, this quality of wear leveling can be implemented separately from any file system. For example, the file system will not jeopardize the described wear leveling even if it writes only to a single virtual address. This may allow the simplification of flash file system code. For instance, a file used as a ring buffer may write to the same virtual block addresses over and over, while most files on the disk are not touched. This, and any other seemingly hostile pattern, should not cause premature device wear using the principles described in this document. With a maximum endurance of 10,000 cycles, a 98% efficiency is still achieved.

References:

[1] Patent Publication: US20100174845A1. Wear leveling for non-volatile memories: maintenance of experience count and passive techniques. Priority Date: January 5, 2009.

[2] Jung, D., Chae, Y., Jo, H., Kim, J., & Lee, J. 2007. A group-based wear-leveling algorithm for large-capacity flash memory storage systems. Proceedings of the 2007 International Conference on Compilers, Architecture, and Synthesis for Embedded Systems. CASES '07, September 30–October 3, 2007, Salzburg, Austria, 5 pages. doi: http://dx.doi.org/10.1145/1289881.1289911.