

DUPLICATE-COPY CACHE USING HETEROGENEOUS MEMORY TYPES

TECHNICAL FIELD

[0001] This invention relates to systems and methods for implementing a cache made up of heterogeneous memory types.

BACKGROUND

[0002] In the field of computing, a “cache” typically refers to a small, fast memory or storage device used to store data or instructions that were accessed recently, are accessed frequently, or are likely to be accessed in the future. Reading from or writing to a cache is typically cheaper (in terms of access time and/or resource utilization) than accessing other memory or storage devices. Once data is stored in cache, it can be accessed in cache instead of re-fetching and/or re-computing the data, saving both time and resources.

[0003] Caches are often provided as multi-level caches. For example, a caching system may include both a “primary” and “secondary” caches. When reading data, a computing system or device may first look for data in the primary cache and, if the data is absent, look for the data in the secondary cache. If the data is not in either cache, the computing system or device may retrieve the data from disk drives or other backend storage devices that reside behind the cache. When writing data, a computing system or device may write data to the primary cache. This data may eventually be destaged to the secondary cache or a storage device to make room in the primary cache.

[0004] Flash memory and other solid-state memory devices can potentially create caches with much larger storage capacities than those using more expensive memory such as dynamic random-access memory (DRAM) cache. For example, storage class memory (SCM), a type of non-volatile NAND flash memory, provides access speeds that are much higher than solid state drives (SSDs). SCM is much cheaper than DRAM but has higher latency than DRAM (microseconds compared to nanoseconds). Because SCM uses flash memory to store data, SCM exhibits some of the same limitations and deficiencies as flash memory, such as write-cycle limits and issues with data fragmentation.

SUMMARY

[0005] The invention has been developed in response to the present state of the art and, in particular, in response to the problems and needs in the art that have not yet been fully solved by currently available systems and methods. Accordingly, systems and methods have been developed to implement a cache using heterogeneous

memory types. The features and advantages of the invention will become more fully apparent from the following description and appended claims, or may be learned by practice of the invention as set forth hereinafter.

[0006] Consistent with the foregoing, a method for demoting data from a cache comprising heterogeneous memory types is disclosed. The method maintains, for a data element in the cache, a write access count that is incremented each time the data element is updated in the cache. The cache includes a higher performance portion and a lower performance portion. The method removes the data element from the higher performance portion in accordance with a cache demotion algorithm. If the data element also resides in the lower performance portion and the write access count is below a first threshold, the method leaves the data element in the lower performance portion. If the data element also resides in the lower performance portion and the write access count is at or above the first threshold, the method removes the data element from the lower performance portion.

[0007] A corresponding system and computer program product are also disclosed and claimed herein.

BRIEF DESCRIPTION OF THE DRAWINGS

[0008] In order that the advantages of the invention will be readily understood, a more particular description of the invention briefly described above will be rendered by reference to specific embodiments illustrated in the appended drawings. Understanding that these drawings depict only typical embodiments of the invention and are not therefore to be considered limiting of its scope, the embodiments of the invention will be described and explained with additional specificity and detail through use of the accompanying drawings, in which:

[0009] Figure 1 is a high-level block diagram showing one example of a network environment in which systems and methods in accordance with the invention may be implemented;

[0010] Figure 2 is a high-level block diagram showing one example of a storage system for use in the network environment of Figure 1;

[0011] Figure 3 is a high-level block diagram showing a cache comprising a higher performance portion (e.g., DRAM cache) and a lower performance portion (e.g., SCM cache);

[0012] Figure 4 is a high-level block diagram showing a cache management module in accordance with the invention;

[0013] Figure 5 is a flow diagram showing one embodiment of a method that may be executed in response to a read hit in the higher performance portion;

[0014] Figure 6 is a flow diagram showing one embodiment of a method that may be executed in response to a read hit in the lower performance portion;

[0015] Figure 7 is a flow diagram showing an alternative embodiment of a method that may be executed in response to a read hit in the lower performance portion;

[0016] Figure 8 is a flow diagram showing one embodiment of a method that may be executed in response to a read miss;

[0017] Figure 9 is a flow diagram showing one embodiment of a method that may be executed in response to a write;

[0018] Figure 10 is a flow diagram showing one embodiment of a method that may be executed when a data element is demoted from the higher performance portion;

[0019] Figure 11 is a flow diagram showing one embodiment of a method that may be executed when a data element is demoted from the lower performance portion;

[0020] Figure 12 is a flow diagram showing an alternative embodiment of a method that may be executed in response to a read hit in the lower performance portion; and

[0021] Figure 13 is a flow diagram showing an alternative embodiment of a method that may be executed when a data element is demoted from the higher performance portion.

DETAILED DESCRIPTION

[0022] It will be readily understood that the components of the present invention, as generally described and illustrated in the Figures herein, could be arranged and designed in a wide variety of different configurations. Thus, the following more detailed description of the embodiments of the invention, as represented in the Figures, is not intended to limit the scope of the invention, as claimed, but is merely representative of certain examples of presently contemplated embodiments in accordance with the invention. The presently described embodiments will be best understood by reference to the drawings, wherein like parts are designated by like numerals throughout.

[0023] The present invention may be embodied as a system, method, and/or computer program product. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present invention.

[0024] The computer readable storage medium may be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as

being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[0025] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[0026] Computer readable program instructions for carrying out operations of the present invention may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++ or the like, and conventional procedural programming languages, such as the "C" programming language or similar programming languages.

[0027] The computer readable program instructions may execute entirely on a user's computer, partly on a user's computer, as a stand-alone software package, partly on a user's computer and partly on a remote computer, or entirely on a remote computer or server. In the latter scenario, a remote computer may be connected to a user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instructions by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present invention.

[0028] Aspects of the present invention are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the invention. 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, may be implemented by computer readable program instructions.

[0029] These computer readable program instructions 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 instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[0030] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus, or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0031] Referring to Figure 1, one example of a network environment 100 is illustrated. The network environment 100 is presented to show one example of an environment where systems and methods in accordance with the invention may be implemented. The network environment 100 is presented by way of example and not limitation. Indeed, the systems and methods disclosed herein may be applicable to a wide variety of different network environments, in addition to the network environment 100 shown.

[0032] As shown, the network environment 100 includes one or more computers 102, 106 interconnected by a network 104. The network 104 may include, for example, a local-area-network (LAN) 104, a wide-area-network (WAN) 104, the Internet 104, an intranet 104, or the like. In certain embodiments, the computers 102, 106 may include both client computers 102 and server computers 106 (also referred to herein as "host systems" 106). In general, the client computers 102 initiate communication sessions, whereas the server computers 106 wait for requests from the client computers 102. In certain embodiments, the computers 102 and/or servers 106 may connect to one or more internal or external direct-attached storage systems 110a (e.g., arrays of hard-disk drives, solid-state drives, tape drives, etc.). These computers 102, 106 and direct-attached storage systems 110a may communicate using protocols such as ATA, SATA, SCSI, SAS, Fibre Channel, or the like.

[0033] The network environment 100 may, in certain embodiments, include a storage network 108 behind the servers 106, such as a storage-area-network (SAN) 108 or a LAN 108 (e.g., when using network-attached storage). This network 108 may connect the servers 106 to one or more storage systems, such as arrays 110b of hard-disk drives or solid-state drives, tape libraries 110c, individual hard-disk drives 110d or solid-state drives 110d, tape

drives 110e, CD-ROM libraries, or the like. To access a storage system 110, a host system 106 may communicate over physical connections from one or more ports on the host system 106 to one or more ports on the storage system 110. A connection may be through a switch, fabric, direct connection, or the like. In certain embodiments, the servers 106 and storage systems 110 may communicate using a networking standard such as Fibre Channel (FC).

[0034] Referring to Figure 2, one embodiment of a storage system 110 containing an array of hard-disk drives 204 and/or solid-state drives 204 is illustrated. As shown, the storage system 110 includes a storage controller 200, one or more switches 202, and one or more storage drives 204, such as hard disk drives 204 or solid-state drives 204 (such as flash-memory-based drives 204). The storage controller 200 may enable one or more host systems 106 (e.g., open system and/or mainframe servers 106 running operating systems such as z/OS, zVM, or the like) to access data in the one or more storage drives 204.

[0035] In selected embodiments, the storage controller 200 includes one or more servers 206. The storage controller 200 may also include host adapters 208 and device adapters 210 to connect the storage controller 200 to host systems 106 and storage drives 204, respectively. Multiple servers 206a, 206b may provide redundancy to ensure that data is always available to connected host systems 106. Thus, when one server 206a fails, the other server 206b may pick up the I/O load of the failed server 206a to ensure that I/O is able to continue between the host systems 106 and the storage drives 204. This process may be referred to as a “failover”.

[0036] In selected embodiments, each server 206 may include one or more processors 212 and memory 214. The memory 214 may include volatile memory (e.g., RAM) as well as non-volatile memory (e.g., ROM, EPROM, EEPROM, hard disks, flash memory, etc.). The volatile and non-volatile memory may, in certain embodiments, store software modules that run on the processor(s) 212 and are used to access data in the storage drives 204. These software modules may manage all read and write requests to logical volumes in the storage drives 204.

[0037] In selected embodiments, the memory 214 includes a cache 218, such as a DRAM cache 218. Whenever a host system 106 (e.g., an open system or mainframe server 106) performs a read operation, the server 206 that performs the read may fetch data from the storage drives 204 and save it in its cache 218 in the event it is required again. If the data is requested again by a host system 106, the server 206 may fetch the data from the cache 218 instead of fetching it from the storage drives 204, saving both time and resources. Similarly, when a host system 106 performs a write, the server 106 that receives the write request may store the write in its cache 218, and destage the write to the storage drives 204 at a later time. When a write is stored in cache 218, the write may also be stored in non-volatile storage (NVS) 220 of the opposite server 206 so that the write can be recovered by the opposite server 206 in the event the first server 206 fails. In certain embodiments, the NVS 220 is implemented as battery-backed memory in the opposite server 206.

[0038] One example of a storage system 110 having an architecture similar to that illustrated in Figure 2 is the IBM DS8000™ enterprise storage system. The DS8000™ is a high-performance, high-capacity storage controller providing disk storage that is designed to support continuous operations. Nevertheless, the systems and methods disclosed herein are not limited to operation with the IBM DS8000™ enterprise storage system 110, but may operate with any comparable or analogous storage system 110, regardless of the manufacturer, product name, or components or component names associated with the system 110. Furthermore, any storage system that could benefit from one or more embodiments of the invention is deemed to fall within the scope of the invention. Thus, the IBM DS8000™ is presented by way of example and is not intended to be limiting.

[0039] Referring to Figure 3, as previously mentioned, flash memory and other solid-state memory devices can potentially create caches with much larger storage capacities than those using more expensive memory such as DRAM cache. For example, storage class memory (SCM), a type of non-volatile NAND flash memory, provides access speeds that are much higher than solid state drives (SSDs). SCM is much cheaper than DRAM but has higher latency than DRAM (microseconds compared to nanoseconds). Because SCM may use flash memory to store data, SCM may exhibit some of the same limitations and deficiencies as flash memory, such as write-cycle limits and issues with data fragmentation. Because of its potential to create caches with much larger storage capacities, systems and methods are needed to effectively incorporate flash memory, such as SCM, into cache. Ideally, such systems and method will take into account the limitations and deficiencies of flash memory, such as write-cycle limits and data fragmentation issues.

[0040] Figure 3 is a high-level block diagram showing a heterogeneous cache 218 (i.e., a cache 218 made up of heterogeneous memory types) comprising a higher performance portion 218a and a lower performance portion 218b. In certain embodiments, the higher performance portion 218a is made up of DRAM memory and the lower performance portion 218b is made up of SCM memory, although neither are limited to these types of memory. The higher performance portion 218a and lower performance portion 218b may be used together to provide a cache 218 within a storage system 110 such as the IBM DS8000™ enterprise storage system. Because memory making up the lower performance portion 218b is likely cheaper than memory making up the higher performance portion 218a, the lower performance portion 218b may be larger, perhaps much larger, than the higher performance portion 218a.

[0041] As shown, the higher performance portion 218a may have associated therewith a cache directory 300a, an LRU (least recently used) list 302a, and statistics 304a. The cache directory 300a may record which data elements are stored in the higher performance portion 218a and where they are stored. The LRU list 302a may be used to determine which data element in the higher performance portion 218a is the least recently used. The statistics 304 may include a read access count 306a and a write access count 308a for each data element (e.g., track) that resides in the higher performance portion 218a. The read access count 306 may be incremented each time the

data element is read in the higher performance portion 218a. The write access count 308 may be incremented each time the data element is modified in the higher performance portion 218a.

[0042] Similarly, the lower performance portion 218b may also include a cache directory 300b, LRU list 302b, and statistics 304b. The cache directory 300b may record which data elements are stored in the lower performance portion 218b and where they are stored. The LRU list 302b may be used to determine which data element is the least recently used in the lower performance portion 218b. The statistics 304b may include a read access count 306b and write access count 308b for each data element (e.g., track) in the lower performance portion 218b. The read access count 306b may be incremented each time the corresponding data element is read in the lower performance portion 218b. The write access count 308b may be incremented each time the corresponding data element is modified in the lower performance portion 218b.

[0043] Referring to Figure 4, in certain embodiments, a cache management module 400 may be used to manage a heterogeneous cache 218 such as that illustrated in Figure 3. Such a cache management module 400 may be hosted within the storage controller 200. The cache management module 400 may include various sub-modules to provide various features and functions. These modules may be implemented in hardware, software, firmware, or combinations thereof. The cache management module 400 and associated sub-modules are presented by way of example and not limitation. More or fewer sub-modules may be provided in different embodiments. For example, the functionality of some sub-modules may be combined into a single or smaller number of sub-modules, or the functionality of a single sub-module may be distributed across several sub-modules.

[0044] As shown, the cache management module 400 includes one or more of a statistics update module 402, read hit module 404, read miss module 406, write module 408, and demotion module 410. The statistics update module 402 maintains the statistics 304 associated with the heterogeneous cache 218. For example, each time a data element is read in the heterogeneous cache 218, the statistics update module 402 updates the associated read access count 306. Similarly, each time a data element is updated in the heterogeneous cache 218, the statistics update module 402 updates the associated write access count 308.

[0045] The read hit module 404 performs various actions when a read hit occurs in the heterogeneous cache 218. This may include either a read hit in the higher performance portion 218a or a read hit in the lower performance portion 218b. Several methods 500, 600, 700 that may be executed by the read hit module 404 will be discussed in association with Figures 5 through 7. By contrast, the read miss module 406 may perform various actions when a read miss occurs in the heterogeneous cache 218. One embodiment of a method 800 that may be executed by the read miss module 406 will be discussed in association with Figure 8.

[0046] The write module 408 may perform various actions when data elements are updated in the heterogeneous cache 218. One embodiment of a method 900 that may be executed by the write module 408 will be discussed in association with Figure 9. The demotion module 410, by contrast, may perform actions associated with demoting data elements from the heterogeneous cache 218 in order to clear storage space in the heterogeneous cache 218. Various methods 1000, 1100 that may be executed by the demotion module 410 will be discussed in association with Figures 10 and 11.

[0047] Referring to Figure 5, one embodiment of a method 500 that may be executed in response to a read hit in the higher performance portion 218a of the heterogeneous cache 218 is illustrated. As shown, the method 500 determines 502 whether a read hit occurred in the higher performance portion 218a. That is, the method 500 determines 502 whether, in response to a read I/O request, a data element associated with the read I/O request was found in the higher performance portion 218a. If so, the method 500 retrieves 504 and returns 504 the data element from the higher performance portion 218a to an originator of the read request. The method 500 increments 506 the read access count 306a associated with the data element in response to the read hit.

[0048] Referring to Figure 6, one embodiment of a method 600 that may be executed in response to a read hit in the lower performance portion 218b is illustrated. As shown, the method 600 determines 602 whether a read hit occurred in the lower performance portion 218b of the heterogeneous cache 218. If so, the method 600 retrieves 604 and returns 604 a data element associated with the read request from the lower performance portion 218b to an originator of the read request. The method 600 then increments 606 the read access count 306b associated with the data element.

[0049] In response to the read hit, the method 600 also allocates 608 space in the higher performance portion 218a that is sufficient to accommodate the data element. In certain embodiments, this may include clearing space in the higher performance portion 218a by demoting a least recently used data element from the higher performance portion 218a. The method 600 then copies 610 the data element associated with the read request from the lower performance portion 218b to the higher performance portion 218a. The data element may then be removed 612 from the lower performance portion 218b. This may leave a single copy of the data element in the higher performance portion 218a.

[0050] Referring to Figure 7, an alternative embodiment of a method 700 that may be executed in response to a read hit in the lower performance portion 218b is illustrated. This method 700 may be executed in place of the method 600 of Figure 6. Each of the steps 702, 704, 706, 708, 710, are similar or identical to those disclosed in Figure 6 except for the final step 712. In step 712, the alternative method 700 removes 712 the data element from the lower performance portion 218b only if the read access count 306b associated with the data element is below a selected threshold. This step 712 preserves a second copy of the data element in the lower performance portion

218b if the data element is read frequently. In the event the data element is demoted from the higher performance portion 218a at a future point in time, a copy of the data element will still reside in the lower performance portion 218b. This eliminates or reduces the need to copy the data element from the higher performance portion 218a to the lower performance portion 218b when the data element is demoted, which reduces processor utilization.

[0051] Referring to Figure 8, one embodiment of a method 800 that may be executed in response to a read miss is illustrated. A read miss may occur when a requested data element cannot be found in either the higher performance portion 218a or the lower performance portion 218b of the heterogeneous cache 218. As shown, if a read miss occurs at step 802, the method 800 brings 804 the data element into the higher performance portion 218 of the heterogeneous cache 218 from backend storage drives 204. The method 800 does not place 806 the data element in the lower performance portion 218b at this time.

[0052] Referring to Figure 9, one embodiment of a method 900 that may be executed in response to updating a data element in the heterogeneous cache 218 is illustrated. As shown, the method 900 determines 902 whether a write to a data element is requested. If so, the method 900 writes 904 the data element to the higher performance portion 218a of the heterogeneous cache 218. The method 900 also writes 906 the data element to the NVS 220 for purposes of redundancy as was previously described.

[0053] At this point, the method 900 determines 908 whether the data element (or a previous version thereof) is contained in the lower performance portion 218b. If the data element is stored in the lower performance portion 218b, the method 900 copies 910 statistics 304b (i.e., the read access count 306b and write access count 308b) associated with the data element from the lower performance portion 218b to the higher performance portion 218a. The method 900 then removes 910 the data element from the lower performance portion 218b.

[0054] Referring to Figure 10, one embodiment of a method 1000 that may be executed when a data element is demoted (i.e., evicted) from the higher performance portion 218a is illustrated. Such a demotion may occur when space is needed in the higher performance portion 218a to accommodate additional data elements. As shown, the method 1000 initially determines 1002 whether data needs to be demoted from the higher performance portion 218a. If so, the method 1000 analyzes the LRU list 302a associated with the higher performance portion 218a to determine 1004 the least recently used data element that is in line to be removed from the higher performance portion 218a. The method 1000 then removes 1004 this data element from the higher performance portion 218a.

[0055] At this point, the method 1000 determines 1006 whether the data element that was removed from the higher performance portion 218a is sequential data. If so, nothing further is performed since it would be disadvantageous to add sequential data to the lower performance portion 218b. If the data element is not sequential, the method 1000 determines 1008 whether the read access count 306a associated with the data

element is greater than a specified threshold and determines 1010 whether the write access count 308a associated with the data element is less than a specified threshold. If both of these conditions are true, the method 1000 places 1012 the data element that was demoted from the higher performance portion 218a in the lower performance portion 218b. In essence, the method 1000 places 1012 data elements in the lower performance portion 218b if the data elements are read frequently (thereby enhancing future read performance for the data elements) but written to infrequently since excessive writes to the data elements may place excessive wear on the lower performance portion 218b.

[0056] Referring to Figure 11, one embodiment of a method 1100 that may be executed when a data element is demoted (i.e., evicted) from the lower performance portion 218b is illustrated. Such a demotion may occur when space is needed in the lower performance portion 218b. As shown, the method 1100 initially determines 1102 whether data needs to be demoted from the lower performance portion 218b. If so, the method 1100 analyzes the LRU list 302b associated with the lower performance portion 218b to determine 1103 which data element is in line to be removed from the lower performance portion 218b.

[0057] At this point, the method 1100 determines 1104 whether the read access count 306b for the data element that is line to be removed is greater than a threshold (to determine if the data element is read frequently). If the read access count 306b is above the threshold, the method 1100 leaves 1106 the data element in the lower performance portion 218b and moves a reference to the data element to the MRU (most recently used) end of the LRU list 302b. In other words, the method 1100 does not remove the data element from the lower performance portion 218b since it is determined to be read frequently and would benefit from being retained in the lower performance portion 218b. The method 1100 also resets 1108 (e.g., sets to zero) the read access count 306b associated with the data element. The method 1100 then returns to the top (i.e., step 1102) where it may analyze the next data element in line to be removed from the lower performance portion 218b.

[0058] On the other hand, if, at step 1104, the read access count 306b associated with the data element is not above the threshold, the method 1100 removes 1110 the data element from the lower performance portion 218b.

[0059] Referring generally to Figures 12 and 13, as was previously mentioned, the methods illustrated in Figures 5 through 11 are generally configured to maintain a single copy of a data element in the heterogeneous cache 218. That is, if a data element resides in the higher performance portion 218a, it will generally not reside in (or will be removed from) the lower performance portion 218b, and vice versa. This provides the best overall cache capacity at the expense of having to copy data elements from the higher performance portion 218a to the lower performance portion 218b when data elements are demoted from the higher performance portion 218a, which may increase processor utilization. Thus, a tradeoff exists between cache capacity and processor utilization.

[0060] In certain embodiments, the methods illustrated in Figures 5 through 11 may be modified to maintain, for the most part, a copy of a data element in both the higher performance portion 218a and the lower performance portion 218b. Although this may reduce overall cache storage capacity, it may advantageously reduce processor utilization by reducing the need to copy data elements between the higher performance portion 218a and the lower performance portion 218b.

[0061] Figure 12 is a flow diagram showing an alternative method 1200 that may be executed in response to a read hit in the lower performance portion 218b. This method 1200 may be executed in environments where duplicate copies of data elements are maintained in the higher performance portion 218a and the lower performance portion 218b. This method 1200 may be used to replace the method 600 described in Figure 6, which may be used in environments where only a single copy is maintained.

[0062] As shown, the method 1200 determines 1202 whether a read hit has occurred in the lower performance portion 218b. If so, the method 1200 retrieves 1204 and returns 1206 a data element associated with the read request from the lower performance portion 218b to an originator of the read request. The method 1200 increments 1206 the read access count 306b associated with the data element.

[0063] In response to the read hit, the method 1200 allocates 1208 space in the higher performance portion 218a that is sufficient to accommodate the data element. The method 1200 then copies 1210 the data element associated with the read request from the lower performance portion 218b to the higher performance portion 218a. The data element may be left 1212 in the lower performance portion 218b. This may provide duplicate copies of the data element in the higher performance portion 218a and the lower performance portion 218b.

[0064] Figure 13 is a flow diagram showing an alternative method 1300 that may be executed when a data element is demoted from the higher performance portion 218a. This method 1300 may also be executed in environments where duplicate copies of data elements are maintained in the higher performance portion 218a and the lower performance portion 218b. This method 1300 may be used to replace the method 1000 described in Figure 10, which may be used in environments where only a single copy is maintained.

[0065] As shown, the method 1300 initially determines 1302 whether data needs to be demoted from the higher performance portion 218a. If so, the method 1300 analyzes the LRU list 302a associated with the higher performance portion 218a to determine 1304 which data element is in line to be removed from the higher performance portion 218a. The method 1300 then removes 1304 this data element from the higher performance portion 218a.

[0066] The method 1300 then determines 1306 whether the data element that was removed from the higher performance portion 218a also resides in the lower performance portion 218b. If so, the method 1300 determines 1308 whether a write access count 308b associated with the data element is below a threshold. In essence, this step 1308 determines whether the data element is updated frequently. If the data element is not updated frequently (i.e., the write access count 308b for the data element is below the threshold), the data element is left 1310 in the lower performance portion 218b since the data element will not place excessive wear on the lower performance portion 218b. On the other hand, if the data element is updated frequently (i.e., the write access count 308b for the data element is above the threshold), the method 1300 removes 1312 the data element from the lower performance portion 218b.

[0067] If, at step 1306, the data element is not in the lower performance portion 218b, the method 1300 determines 1314 whether the data element that was removed from the higher performance portion 218a is sequential data. If so, nothing is changed since it would not be advantageous to add the data element to the lower performance portion 218b. If the data element is not sequential, the method 1300 determines 1316 whether a read access count 306b associated with the data element is greater than a specified threshold and determines 1318 whether a write access count 308b associated with the data element is less than a specified threshold. If so, the method 1300 places 1320 the data element that was demoted from the higher performance portion 218a in the lower performance portion 218b. In essence, the method 1300 places 1320 demoted data elements in the lower performance portion 218b if the data elements are read frequently (thereby enhancing future read performance for the data elements) but not updated frequently since excessive writes may place excessive wear on the lower performance portion 218b if it is implemented in a memory type (e.g., SCM) having write cycle limits.

[0068] The flowcharts and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present invention. In this regard, each block in the flowcharts or block diagrams may represent a module, segment, or portion of code, which comprises one or more executable instructions for implementing the specified 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. Other implementations may not require all of the disclosed steps to achieve the desired functionality. It will also be noted that each block of the block diagrams and/or flowchart illustrations, and combinations of blocks in the block diagrams and/or flowchart illustrations, may be implemented by special purpose hardware-based systems that perform the specified functions or acts, or combinations of special purpose hardware and computer instructions.

CLAIMS

1. A method for demoting data from a cache comprising heterogeneous memory types, the method comprising:

maintaining, for a data element, a write access count that is incremented each time a data element is updated in a cache, the cache comprising a higher performance portion and a lower performance portion;
storing the data element in both the higher performance portion and the lower performance portion;
removing the data element from the higher performance portion in accordance with a cache demotion algorithm;

upon removing the data element from the higher performance portion, determining if the write access count is below a first threshold;

in response to removing the data element from the higher performance portion and determining that the write access count is below the first threshold, leaving the data element in the lower performance portion; and

in response to removing the data element from the higher performance portion and determining that the write access count is at or above the first threshold, removing the data element from the lower performance portion.

2. The method of claim 1, wherein the data element is a track.

3. The method of claim 1, wherein removing the data element from the higher performance portion comprises removing the data element from the higher performance portion when the higher performance portion is full.

4. The method of claim 1, wherein the higher performance portion is made up of dynamic random access memory (DRAM) and the lower performance portion is made up of storage class memory (SCM).

5. The method of claim 1, wherein the lower performance portion has a larger storage capacity than the higher performance portion.

6. A computer program product for demoting data from a cache comprising heterogeneous memory types, the computer program product comprising a non-transitory computer-readable storage medium having computer-usable program code embodied therein, the computer-usable program code configured to perform the following when executed by at least one processor:

maintain, for a data element, a write access count that is incremented each time a data element is updated in a cache, the cache comprising a higher performance portion and a lower performance portion;
 store the data element in both the higher performance portion and the lower performance portion;
 remove the data element from the higher performance portion in accordance with a cache demotion algorithm;

upon removing the data element from the higher performance portion, determine if the write access count is below a first threshold;

in response to removing the data element from the higher performance portion and determining that the write access count is below the first threshold, leave the data element in the lower performance portion; and

in response to removing the data element from the higher performance portion and determining that the write access count is at or above the first threshold, remove the data element from the lower performance portion.

7. The computer program product of claim 6, wherein the data element is a track.

8. The computer program product of claim 6, wherein removing the data element from the higher performance portion comprises removing the data element from the higher performance portion when the higher performance portion is full.

9. The computer program product of claim 6, wherein the higher performance portion is made up of dynamic random access memory (DRAM) and the lower performance portion is made up of storage class memory (SCM).

10. The computer program product of claim 6, wherein the lower performance portion has a larger storage capacity than the higher performance portion.

11. A system for demoting data from a cache comprising heterogeneous memory types, the system comprising:

at least one processor; and

at least one memory device operably coupled to the at least one processor and storing instructions for execution on the at least one processor, the instructions causing the at least one processor to:

maintain, for a data element, a write access count that is incremented each time a data element is updated in a cache, the cache comprising a higher performance portion and a lower performance portion;

store the data element in both the higher performance portion and the lower performance portion;

remove the data element from the higher performance portion in accordance with a cache demotion algorithm;

upon removing the data element from the higher performance portion, determine if the write access count is below a first threshold;

in response to removing the data element from the higher performance portion and determining that the write access count is below the first threshold, leave the data element in the lower performance portion; and

in response to removing the data element from the higher performance portion and determining that the write access count is at or above the first threshold, remove the data element from the lower performance portion.

12. The system of claim 11, wherein removing the data element from the higher performance portion comprises removing the data element from the higher performance portion when the higher performance portion is full.

13. The system of claim 11, wherein the higher performance portion is made up of dynamic random access memory (DRAM) and the lower performance portion is made up of storage class memory (SCM).

14. The system of claim 11, wherein the lower performance portion has a larger storage capacity than the higher performance portion.