Title: CONFIGURABLE AND SCALABLE HYBRID MULTI-TIERED CACHING STORAGE SYSTEM

Abstract: A hybrid storage system comprising mechanical disk drive means, flash memory means, SDRAM memory means, and SRAM memory means is described. IO processor circuits and DMA controller circuits are devised to eliminate host intervention. Multi-tiered caching system and novel data structure for mapping logical address to physical address results in a configurable and scalable high performance computer data storage solution.
Published:
— without international search report and to be republished upon receipt of that report

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.
SPECIFICATION

Title

Configurable and Scalable Hybrid Multi-Tiered Caching Storage System

Cross-Reference to Related Applications

This application claims the benefit of U.S. Non-Provisional application having serial number 11/450,023, filed 8 June 2006, and entitled "Configurable and Scalable Hybrid Multi-Tiered Caching Storage System using SRAM, SDRAM, Flash and Mechanical Hard Disk Drives"; and of U.S. Non-Provisional application having serial number 11/450,005, filed 8 June 2006, and entitled "Optimized Placement Policy for Solid State Storage Devices.

Background

(1) Technical Field

The present invention relates to a data storage system which is applied to a computer system, and comprises volatile (e.g. SRAM, SDRAM) and nonvolatile (e.g. flash memory, mechanical hard disk) storage components.

(2) Background Art

In a conventional computer system, a hard disk drive (HDD) is used as an external memory device wherein a magnetic disk is used as a storage medium. The HDD can be used as a large-capacity file apparatus. However, as compared to a main memory comprising a semiconductor memory (e.g. a DRAM), the access speed of the HDD is lower. A cache system for the HDD has been known as means for increasing the access speed of the HDD. Dynamic random access memory (DRAM) and flash memory has been used to implement a cache system for the HDD. However, the translation from logical address to physical address format suitable for accessing flash memory and HDD consume resources of the host computer and affects performance. Accordingly there is a need for a hybrid storage system wherein the performance is improved through elimination of host intervention.
Enterprise-level storage systems typically use arrays of hard disk drives (HDD) as mass storage units, or configured as RAID systems. Data users or clients access the data using standard block-based IO interfaces or over the network using standard file-based access protocols. The HDD array data is managed by dedicated host computers that run storage management applications. As several interface controllers are employed in both host and client systems, enterprise-level storage systems will benefit from a controller architecture that integrates block-based access and file-based or random access to the data. The integration of data transfer controllers for different interfaces in the previously mentioned hybrid storage system that implements multi-tiered caching system for a HDD will extend the benefits of HDD data caching to HDD array systems.
Summary

A hybrid storage system solution that includes a variety of storage devices, such as a mechanical disk drive, a flash memory, a SDRAM memory, and a SRAM memory is described. The hybrid storage solution includes an IO processor and multiple DMA controllers so that host intervention may be minimized or avoided. In addition, a multi-tiered caching system and novel data structures for mapping logical address to physical address may also be used, resulting in a configurable and scalable high performance computer data storage solution.

A LBA_FLASH_HDD table has a first portion for mapping logical address to flash address and a second portion for mapping a logical address to a disk drive address. The LBA_FLASH_HHD table may be stored in several places. For example, a copy may be stored in a non-volatile memory, such as the flash memory, a most frequently used portion is stored in SRAM and a remainder is stored in SDRAM. In addition, a back-up copy of the table may also be stored in the mechanical disk drive. A LBA_SDRAM table may be used to map a logical address to SRAM and SDRAM addresses. This table may be stored in SDRAM and cached in SRAM. Several DMA controllers are provided for moving data among multi-tiered storage devices. The IO processor may be used to implement a uniform method for DMA by preparing DMA instructions, which may be in a linked list format.

In one embodiment of the present invention, an IO processor, DMA controllers, and all necessary control functions are integrated in a SOC device. At least eight configuration examples of the storage solution are disclosed. In the first example, the storage system SOC device is configured as a slave device and interfaces with the host system through a system bus that is capable of random access and DMA such as PCI/PCI-X/PCI Express, and also interfaces with mechanical disk drives through a standard IO storage interface such as ATA or SCSI.
In the second example, the storage system SOC device is configured as a host system that interfaces with an external storage device through a system bus that is capable of random access and DMA such as PCI/PCI-X/PCI Express.

In the third example, the storage system SOC device is configured as a standalone host system that interfaces with mechanical disk drives through a standard IO storage interface such as SCSI, and also interfaces to a network through a standard IO network interface such as Ethernet.

In the fourth example, the storage system SOC device is configured as a slave device and interfaces with the host system through an internal standard IO such as Fiber Channel and interfaces with mechanical disk drives through a second standard IO interface such as USB.

In the fifth example, the storage system SOC device is configured as a host system that interfaces with an external storage device through an external standard IO storage interface such as Fiber Channel, and also interfaces to a network through a standard IO network interface such as Ethernet.

In the sixth example, the storage system SOC device is configured as a slave device and interfaces with the host system through an external standard IO interface such as Fiber Channel and interfaces with mechanical disk drives through a second standard IO interface such as USB.

In the seventh example, the storage system SOC device is configured as a host system that interfaces with an external storage device through an internal standard IO storage interface such as Fiber Channel, and also interfaces to a network through a standard IO network interface such as Ethernet.
In the eighth example, the storage system SOC device is configured as a slave device and integrated into a mechanical disk drive in a single enclosure and interfaces with the host system through a standard IO interface such as SCSI and interfaces with the magnetic disk controller through low level direct connections.
Brief Description of the Drawings

So that the manner in which the above recited features, advantages and objects of the present invention are attained and can be understood in detail, a more particular description of
the invention, briefly summarized above, may be had by reference to the embodiments thereof which are illustrated in the appended drawings.

It is to be noted, however, that the appended drawings illustrate only typical embodiments of this invention and are therefore not to be considered limiting of its scope, for the present invention may admit to other equally effective embodiments.

Figure 1 is a diagram illustrating the components comprising the hybrid storage device according to an embodiment of the present invention.

Figure 2 is a diagram illustrating an example configuration of an embodiment of the present invention where the storage system SOC device is configured as a slave device and interfaces with the host system through a system bus that is capable of random access and DMA such as PCI/PCI-X/PCI Express, and also interfaces with mechanical disk drives through a standard IO storage interface such as ATA or SCSI.

Figure 3 is a diagram illustrating a second example configuration of an embodiment of the present invention where the storage system SOC device is configured as a host system that interfaces with an external storage device through a system bus that is capable of random access and DMA such as PCI/PCI-X/PCI Express.

Figure 4 is a diagram illustrating a third example configuration of an embodiment of the present invention where the storage system SOC device is configured as a standalone host system that interfaces with mechanical disk drives through a standard IO storage interface such as SCSI, and also interfaces to a network through a standard IO network interface such as Ethernet.
Figure 5 is a diagram illustrating a fourth example configuration of an embodiment of the present invention where the storage system SOC device is configured as a slave device and interfaces with the host system through an internal standard IO such as Fiber Channel and interfaces with mechanical disk drives through a second standard IO interface such as USB.

Figure 6 is a diagram illustrating a fifth example configuration of an embodiment of the present invention where the storage system SOC device is configured as a host system that interfaces with an external storage device through an external standard IO storage interface such as Fiber Channel, and also interfaces to a network through a standard IO network interface such as Ethernet.

Figure 7 is a diagram illustrating a sixth example configuration of an embodiment of the present invention where the storage system SOC device is configured as a slave device and interfaces with the host system through an external standard IO interface such as Fiber Channel and interfaces with mechanical disk drives through a second standard IO interface such as USB.

Figure 8 is a diagram illustrating a seventh example configuration of an embodiment of the present invention where the storage system SOC device is configured as a host system that interfaces with an external storage device through an internal standard IO storage interface such as Fiber Channel, and also interfaces to a network through a standard IO network interface such as Ethernet.

Figure 9 is a diagram illustrating an eighth example configuration of an embodiment of the present invention where the storage system SOC device is configured as a slave device and integrated into a mechanical disk drive in a single enclosure and interfaces with the host system through a standard IO interface such as SCSI and interfaces with the magnetic disk controller through low level direct connections.
Figure 10 is a diagram illustrating the data structures inside the nonvolatile and volatile storage components according to an embodiment of the present invention.

Figure 10a is a diagram illustrating an example data structure for LBA-Flash-HDD mapping table according to an embodiment of the present invention.

Figure 10b is a diagram illustrating an example data structure for LBA-SDRAM mapping table according to an embodiment of the present invention.

Figure 11 is a diagram illustrating a power up initialization process flow according to an embodiment of the present invention.

Figure 12 is a diagram illustrating a process flow of a block read command from a host computer system according to an embodiment of the present invention.

Figure 13 is a diagram illustrating a process flow of a block write command from a host computer system according to an embodiment of the present invention.

Figure 14 is a diagram illustrating a process flow of a random byte-addressed read access to the hybrid storage device according to an embodiment of the present invention.

Figure 15 is a diagram illustrating a process flow of a random byte-addressed write access to the hybrid storage according to an embodiment of the present invention.

Figure 16 is a diagram illustrating a process flow of a DMA transfer according to an embodiment of the present invention.

Figure 17 is a diagram illustrating a process flow of the update of the mapping tables in response to a read request according to an embodiment of the present invention.

Figure 18 is a diagram illustrating a process flow for updating the mapping tables in response to a write request according to an embodiment of the present invention.
Figure 19 is a diagram illustrating a process flow for updating the mapping tables during a data flush operation in response to a write request or activated as a background process according to an embodiment of the present invention.
Detailed Description of the Invention

Figure 1 is a diagram illustrating the components comprising the hybrid storage device 101 according to an embodiment of the present invention.

The hybrid storage device 101 comprises several storage devices listed as follows in order of increasing storage capacity and decreasing access time - embedded SRAM 105, array of SDRAM devices 108, array of flash devices 110 and array of hard drives (not shown). Three levels of caching are implemented in the storage system: flash array caches data in hard drives, SDRAM array caches data in flash array, SRAM caches data in SDRAM array. The main non-volatile storage component comprises one or more hard disks (not shown).

Hybrid storage controller 102 is a chip that manages the storage system. It contains multiple embedded DMA controllers:

PCI-Express/PCI-X/PCI DMA controller 111 handles byte or word addressable access to stored data by any device connected via a system bus such as: PCI-Express, PCI-X, PCI interface. Content addressable memory (CAM) 112 stores a look-up table used by the PCI-Express/PCI-X/PCI interface DMA Controller to look-up the block address associated with the byte or word address.

IO Storage DMA Controller 113 and IO Storage DMA Controller 114 handle DMA via standard block access IO interfaces such as: IDE/ATA, serial ATA, USB, SCSI, etc. The said IO DMA Controllers can be used to connect to a host computer system through an IO interface. They can also be used to control arrays of hard disks.

IO Network Interface DMA Controller 115 and IO Network Interface DMA Controller 116 handle DMA to a network interface such as: ETHERNET, USB, FIREWIRE, FIBER CHANNEL and any combination of these network interfaces.

Flash DMA Controller 109 handles DMA to the flash array 110.
SDRAM Controller 107 handles DMA to the SDRAM array 108.

Hybrid storage controller 102 contains an IO processor 103 that can be implemented using one or more embedded processors. The IO processor handles the processing of host commands (read/write) and run the algorithms for managing the different storage media. The implementation of the caching algorithm and maintenance of control structures such as translation tables is transparent to external entities that use or connect to the storage system such as a host computer system.

One or more additional embedded processor(s) 114 can function as compute/application processors 104 running over conventional O/S such as Windows, Linux, etc. The SDRAM 105 can be shared between the IO processor and the application processor(s). Several DMA paths 119,120 are provided to avoid data bottlenecks. The different storage media can be connected in different ways to the DMA paths to achieve the most optimized traffic distribution. For example, at least one of the DMA paths 120 can be used as dedicated path between the SRAM 105 and the flash array 110. During reads to data that is in the flash array, if the traffic is heavy on the other paths, then this DMA path 119 can be used to transfer data from the flash array 110 to a temporary store buffer in the SRAM 105.

Furthermore, separate control paths 118 are provided for the embedded processors to access the register interface of the different DMA controllers, reducing the control overhead on the high-speed DMA paths 119,120. A Field-Programmable ROM 106 can be employed to store boot code for the IO processor.

During normal operation, data and control information is distributed among the storage components, as illustrated in figure 10 and discussed in detail later in this description. A device 117, such as a PowerGuard® circuit available from BiTMICRO Networks, Inc. of Fremont California, ensures that the data in the flash 110, SDRAM 108 and in all the components in the
hybrid storage controller 102 are protected in the event of power loss. In an alternative embodiment, an uninterruptible power supply may be used in lieu of or in addition to a device such as device 117. The data in the volatile SDRAM and SRAM will get flushed to the flash. Thus, the flash retains all cached data and control information. PowerGuard® circuit protects all the components in the hybrid storage controller 101 including the embedded processors and all embedded FIFOs, internal RAMs. On power loss, processing of retained information in these components continues. Transient data from the external entities that connect to the hybrid storage device will not be accepted.

The IO processor 103 instructs the IO Storage Interface DMA controllers 113 and 114, the IO Network Interface DMA controllers 115 and 116, the PCI-Express/PCI-X/PCI DMA controller 111 and the flash DMA controller 109 to transfer data between the SRAM 105 or SDRAM 108 and their respective interfaces. The storage system is managed such that data transferred by the DMA controllers to their respective interfaces are always cached in the SRAM 105 or SDRAM 108, which provides faster access compared to the flash 110 and the hard drives (not shown). The IO processor 103 includes in the instructions such information as the direction of the data transfer, the source and destination addresses, the size of data to be transferred, and all other interface-specific control information. The instructions are stored to the SRAM 105 or SDRAM 108. Each instruction contains a link to the next instruction. Hence, after the IO processor 103 posts an initial instruction via the control bus to any of the DMA controllers, the DMA controller can automatically fetch the next instruction from the SRAM 105 or SDRAM 108. The IO processor 103 is then informed of the completion of a data transfer by the DMA controller. The process flow for performing DMA transfers is illustrated in figure 16 and discussed in detail later in this description.
Figure 2 is a diagram illustrating an example configuration of an embodiment of the present invention. In this configuration, the hybrid storage controller 201 is configured as a slave device and interfaces with the host system 202 through a system bus that is capable of random access and DMA such as PCI/PCI-X/PCI Express. The hybrid storage device controls an array of hard disk drives 204 through a standard IO storage interface such as Serial ATA.

Figure 3 is a diagram illustrating a second example configuration of an embodiment of the present invention. It shows that the hybrid storage controller 301 can also be configured as a Host system controlling a slave device 302 through a system bus 303 such as PCI/PCI-X/PCI Express. In the illustration, the slave device 302 is actually a hard disk array controller. Since the hybrid storage device is itself configurable to function as a hard disk array controller with an interface to a system bus, then two hybrid storage devices could be interconnected using their PCI/PCI-X/PCI-Express DMA controller where one is a slave device to the other.

Figure 4 is a diagram illustrating a third example configuration of an embodiment of the present invention. It shows the hybrid storage controller 401 is configured as a standalone host system that interfaces with a hard disk drive arrays through a standard IO storage interface such as SCSI, and also interfaces to a network through a standard IO network interface such as Ethernet. The IO Storage Interface DMA Controller 402 handles DMA to the hard disk array 404. The IO Storage Interface DMA Controller 403 handles DMA to the hard disk array 405. The IO Network Interface DMA Controller 406 handles the connection to network 408. The IO Network Interface DMA Controller 407 handles the connection to network 409. The embedded IO Processor 410 coordinates the operation of the said DMA controllers. Since the hybrid storage device functions as a standalone system, then other software applications can be run on the additional embedded Compute/Application processor(s) 411.
Figure 5 is a diagram illustrating a fourth example configuration of an embodiment of the present invention. The hybrid storage controller 501 is configured as a slave device and interfaces with the host system 502 through a standard Block-access IO bus 503 such as Fiber Channel and interfaces with hard disk drives through a second standard IO interface such as IDE. The hybrid storage device in this case uses the internal IO Storage Interface DMA controllers 504 and 505 embedded the hybrid storage controller 501 to handle the interface to both the host system 502 and the hard disk drives.

Figure 6 is a diagram illustrating a fifth example configuration of an embodiment of the present invention. The hybrid storage controller 601 is configured as a host system that interfaces to an external storage device 602 through an external standard IO interface controller 603 connected to it via an IO bus such as Fiber Channel 604. The external storage device may also be another hybrid storage device. The hybrid storage controller also interfaces to a network through a standard IO network interface such as Ethernet 605. The hybrid storage controller uses the PCI-Express/PCI-X/PCI DMA Controller 606 to configure and control the operation of the external IO interface controller and to transfer IO commands, data and status information to and from the external storage device through the external IO interface controller. The PCI-Express/PCI-X/PCI DMA Controller has master and slave interfaces such that either the hybrid storage controller or external IO controller may initiate a DMA transaction. To send IO commands to the external storage device where the external IO controller acts as a DMA slave, the hybrid storage controller writes the IO commands to the external IO controller using the hybrid storage controller PCI-Express/PCI-X/PCI DMA master interface 605. The external IO controller establishes the command phase on the IO bus to send the IO command. To send IO commands to the external storage device where the external IO controller acts as a DMA master, the hybrid storage controller indicates the location of the IO
command buffer in SRAM or SDRAM to the external IO controller. The external IO controller reads the IO command from the hybrid storage controller using the hybrid storage controller PCI-Express/PCI-X/PCI DMA slave interface and establishes the command phase on the IO bus to send the IO command to the external storage device. To transfer data to or from the external storage device where the external IO controller acts as a DMA slave, the hybrid storage controller writes to or reads data from the external IO controller when the IO data phase is established on the IO bus using the hybrid storage controller PCI-Express/PCI-X/PCI DMA master interface. To transfer data to or from the external storage device where the external IO controller acts as a DMA master, the hybrid storage controller indicates the data cache buffer location in SRAM or SDRAM to the external IO controller so that when the IO data phase is established on the IO bus, the external IO controller can write data to or read data from the hybrid storage controller using the hybrid storage controller PCI-Express/PCI-X/PCI DMA slave interface. The data is written to or read from the data cache in SRAM or SDRAM. To receive IO status information from the external storage device where the external IO controller acts as a DMA master, the external IO controller interrupts the hybrid storage controller when a status phase is completed on the IO bus so that the hybrid storage controller can read the received IO status information from the external IO controller through the hybrid storage controller PCI-Express/PCI-X/PCI DMA master interface and transfer it to the IO status buffer in SRAM 707 or SDRAM 708. To receive IO status information from the external storage device where the external IO controller acts as a DMA master, the hybrid storage controller initially indicates the IO status buffer location in SRAM or SDRAM to the external IO controller so that when a status phase is completed on the IO bus, the external IO controller can write the received IO status information to the IO status buffer in the hybrid storage controller through the hybrid storage controller PCI-Express/PCI-X/PCI DMA slave interface.
Figure 7 is a diagram illustrating a sixth example configuration of an embodiment of the present invention. The hybrid storage controller 701 is configured as a slave device and interfaces with the host system 702 through an external IO interface controller 703 using a standard IO interface such as Fiber Channel and interfaces with mechanical disk drives through internal standard IO interfaces such as Serial Attached SCSI and Serial ATA 704 705. The hybrid storage controller uses the PCI-Express/PCI-X/PCI DMA Controller 706 to configure and control the operation of the external IO interface controller and to transfer IO commands, data and status information to and from the host system through the external IO interface controller. The PCI-Express/PCI-X/PCI DMA Controller has master and slave interfaces such that either the hybrid storage controller or external IO controller may initiate a DMA transaction. To transfer IO commands from the host where the external IO controller acts as a DMA slave, the external IO controller interrupts the hybrid storage controller when a command phase is completed on the IO bus 709 so that the hybrid storage controller can read the received IO commands from the external IO controller through the hybrid storage controller PCI-Express/PCI-X/PCI DMA master interface and transfer it to the IO command buffer in SRAM 707 or SDRAM 708. To transfer IO commands from the host where the external IO controller acts as a DMA master, the hybrid storage controller initially indicates the IO command buffer location in SRAM or SDRAM to the external IO controller so that when a command phase is completed on the IO bus, the external IO controller can write the received IO command to the IO command buffer in the hybrid storage controller through the hybrid storage controller PCI-Express/PCI-X/PCI DMA slave interface. To transfer data to or from the host where the external IO controller acts as a DMA slave, the hybrid storage controller writes to or reads data from the external IO controller when the IO data phase is established on the IO bus using the hybrid storage controller PCI-Express/PCI-X/PCI DMA.
master interface. To transfer data to or from the host where the external IO controller acts as a DMA master, the hybrid storage controller indicates the data cache buffer location in SRAM or SDRAM to the external IO controller so that when the IO data phase is established on the IO bus, the external IO controller can write data to or read data from the hybrid storage controller using the hybrid storage controller PCI-Express/PCI-X/PCI DMA slave interface. The data is written to or read from the data cache in SRAM or SDRAM. To send IO status information to the host where the external IO controller acts as a DMA slave, the hybrid storage controller writes the IO status information to the external IO controller using the hybrid storage controller PCI-Express/PCI-X/PCI DMA master interface. The external IO controller establishes the IO status phase on the IO bus and sends the IO status information to the host. To send IO status information to the host where the external IO controller acts as a DMA master, the hybrid storage controller indicates the location of the IO status information in SRAM or SDRAM to the external IO controller so that when the IO status phase is established on the IO bus, the external IO controller can read the IO status information from the hybrid storage controller using the hybrid storage controller PCI-Express/PCI-X/PCI DMA slave interface.

Figure 8 is a diagram illustrating a seventh example configuration of an embodiment of the present invention where the hybrid storage controller 801 is configured as a host system that interfaces with an external storage device 802 through an internal standard IO storage interface such as Serial Attached SCSI 803, and also interfaces to a network through a standard IO network interface such as Ethernet 804. The external storage device may also be another hybrid storage device.

Figure 9 is a diagram illustrating an eighth example configuration of an embodiment of the present invention where the hybrid storage controller device 901 is configured as a slave device and integrated into a hybrid hard disk contained within a single disk drive enclosure,
and interfaces with the host system through a standard IO interface such as Serial ATA and interfaces with the magnetic disk controller through low level direct connections.

Figure 10a is a diagram illustrating the data structures in the nonvolatile and volatile memory components of the storage system according to an embodiment of the present invention. Figure 10a illustrates the different storage media and how each are used to store and cache data, code, and other control data structure. Data 1001 are blocks of data stored in a non-volatile memory, such as hard drive 1002. Data 1003 are blocks of data may also be stored in a non-volatile memory, such as hard drive 1004. The flash may also be used to provide non-volatile memory for storing data. The data 1005 in the flash 1007 is cached portion of the data 1001 in the hard drive 1002. The data 1006 in the flash 1007 is cached portion of the data 1003 the hard drive 1004. The SDRAM 1009 provides faster access storage for data compared to flash 1007 and hard drive 1002 and 1004. The data 1008 in the SDRAM 1009 are the cached portions of the data 1005 and 1006 in the flash 1007. These cached portions are most recently read from or written to the storage system 1000 by the host system (not shown). The SRAM 1026 is the fastest-access storage devices that can be used to store data. In the figure, data 1024 in the SRAM 1026 are also cached portions of data 1005 in the flash 1007. Data 1025 in SRAM 1026 are also cached portions of data 1006 in the flash 1007. The data cached in the SRAM 1026 can be treated in the same way as data in cached in the SDRAM 1009. Cached data is assumed to be the most recently or most frequently accessed from the host. However, for random one-time read accesses that do not necessarily qualify to be cached, the SRAM may also be used as a temporary store for such read data. The buffer gets immediately freed once the data is transferred to the host. The storage system is managed such that data transferred by the DMA controllers (not shown) to/from the host system and to/from the hard drives or flash are always cached in the SDRAM or SRAM.
Code 1012 refers to low-level software that runs on the embedded processor. This code implements the algorithms for managing the storage system. "Code:O/S, Apps" 1013 refers to OS kernel and application codes. Optionally, another embedded processor can be used to run applications under a conventional O/S such as Windows, Linux, etc. 1012 and 1013 are stored in flash 1007 or equivalent non-volatile memory. Since these are critical information, back-up copies 1016, 1017 are stored in the hard drives. The FPROM 1015 is another small-capacity non-volatile storage media that can be used to store a small amount of code 1014 that gets loaded on power-up. However, the initial code loaded on power-up could likewise be loaded from the flash 1007. The rest of the codes 1012, 1013 get paged from the flash 1007 to SRAM 1026. The IO processor executes code off the SRAM 1026 unless it is cached in the first level processor's internal cache (not shown). Hence, the SRAM 1026 serves as a second level cache for the IO processor.

LBA-Flash-HDD Tables 1010 are control structures that maintain the mapping of data logical block addresses (LBA) to their physical locations in the flash and in the hard drives. Flash media caches data in the hard drive. Aside from the physical locations, there is also information relating to the state of the data in the flash (if they are modified, if they are in transit, if they are candidates for getting remapped to other flash locations, if they are candidates for getting flushed back to the HDD). The LBA-Flash-HDD tables 1010 are maintained by the IO processor. More details of maintaining the mapping of system logical block address to Flash physical block address and HDD block address is described in United States Patent Application having Application Serial No. 11/450,005, which was filed on 08 June 2006 and is entitled "Optimized Placement Policy for Solid State Storage Devices", hereinafter named the "Patent Application", and which is hereby incorporated by reference as if fully set forth herein. The most frequently accessed portions 1027 are buffered in the
SRAM 1026 which can be accessed the fastest by the IO processor. Less frequently accessed portions 1011 are buffered in the SDRAM 1009. On power-down, these tables 1027, 1011 are consolidated and the updates are flushed back to the table 1010 in the flash 1007 where they are stored. Since these are critical information, a back-up copy 1016 is stored in the hard drives.

LBA-SDRAM Tables 1018, 1019 extend the LBA-Flash-HDD Tables to also include the mapping of data logical block addresses to their locations in the SDRAM, for those data blocks that are cached in the SDRAM. Aside from the SDRAM location, the table also has additional information relating to the state of the cached data blocks (if they are modified, if they are in transit, if they are candidates for getting flushed to the HDD or the flash). The LBA-SDRAM Tables 1018, 1019 are maintained by the IO processor. The most frequently accessed portions 1019 are stored in the SRAM 1026 which can be accessed the fastest by the IO processor. Less frequently accessed portions 1018 are stored in the SDRAM 1009. Since SDRAM 1009 and SRAM 1026 are volatile storage, LBA-SDRAM tables 1018, 1019 are initially empty, and get built as data blocks get read from or written to the storage system.

Scratch Buffers 1020 is the collective term referring to the temporary storage area that buffers information for the IO processor at run-time e.g. those buffers that queue IO commands for processing, or scratchpad memory used by the OS and applications. Both the SRAM 1026 and SDRAM 1009 can be used to store such information. LBA-SDRAM Tables 1018, 1019 are control information generated only during run-time and are special cases of run-time information that use scratch buffers in the SRAM 1026 and SDRAM 1009.

DMA Instructions 1021 is another set of special case of run-time control information generated by the IO processor. They are generated for use by the DMA controllers. To respond to read/write requests from the host system, the IO processor creates DMA
instructions 1021 for the IO DMA controller or PCI-Express/PCI-X/PCI DMA controller and stores them in the SDRAM 1009 or SRAM 1026. When transferring data blocks to/from the SDRAM to the flash, the IO processor creates DMA instructions for the flash DMA controller. When transferring data to/from the SDRAM to the hard disk drives, the IO processor creates DMA instructions for the IO DMA controller connected to the hard disk drives. A DMA instruction contains links to the next instruction; hence the DMA controllers can automatically fetch the DMA instructions stored in the SDRAM or SRAM. These DMA instructions contain the location in the SDRAM for the DMA controllers to fetch/store data.

ByteAddr-LBA Table 1022 refers to the byte address look-up table used by the PCI-Express/PCI-X/PCI interface DMA controller to look-up the block address associated with the byte or word address. A CAM 1023 is employed for this purpose.

Figure 10b is a diagram illustrating an example data structure for LBA-Flash-HDD mapping table according to an embodiment of the present invention. The actual location of data in the mechanical hard drives as well as cached location in the flash is independently determined by the embedded IO processor without host intervention. For example, if the host uses logical block addresses or LBAs to reference data, such LBAs are translated to physical locations by the IO processor. The IO processor optimizes the physical locations in the hard disks and the flash so that frequently or most recently accessed data are stored in the flash so that they can be accessed in the quickest fashion. An example of such optimization is to distribute a set of LBAs accessed in unison by the host to different devices in the flash array, so that portions of the LBA set can be accessed concurrently. The host accesses can be tracked and the access behavior is used to optimize access performance. -The LBA_Flash_HDD tables refer to the data structures maintained by the embedded processors in order to associate data accesses by the host to their physical locations in the flash and hard drives, and also to allocate
locations in the flash array for those data that are recently or most frequently accessed. It is beneficial to place such data in the flash since transferring data between flash and SDRAM is faster than between hard drives and the SDRAM. Each entry in the table such as 1028 associates a set of LBAs addressed by the host to information regarding their locations in the flash and hard drives.

The information contained in the table is subdivided into the flash remap table 1029 and HDD remap table 1030. The flash remap table 1029 includes information on the physical location (physical block address or PBA 1031) of cached data blocks in the flash array. This particular information is used by the IO processor to build DMA instructions which are interpreted by the flash DMA controller to control the flash device or group of flash devices. Aside from the present physical location, the table also includes information on the caching state 1032 of the data. This information indicates how the cached data in the flash differs compared to its counterpart stored in the hard drives. Such information includes: if that set of data is not yet stored in the hard drives, if they are fully or partly modified, if they are currently in transit and should not be allowed access yet, etc. Lastly, the flash remap table also includes other control information 1033 relating to the usage of the physical flash blocks. Such information determines if the data is a good candidate to get moved to other flash blocks either to prolong the life of the flash, or as part of optimizations to improve the accesses to the data by the host.

The HDD remap 1030 table includes location information (physical block address or PBA 1034) and other control information 1035 such as HDD usage statistics. Location information or PBA 1034 is used by the IO processor to build DMA instructions that are interpreted by the IO DMA controller allowing it to uniquely address the data in the hard drives. IO interfaces such as SCSI or ATA typically use LBA or CHS addressing schemes to
address data in hard drives. The usage statistics are additional information that can be included in control information 1035 and related to the frequency and patterns of usage of the addressed disk sectors or locations. This information can be used by the IO processor in algorithms that optimize the distribution of data to the disks and improve accesses to the data by the host.

Additional details of the optimized method for maintaining the LBA-Flash-HDD mapping table illustrated in Fig 10b are further disclosed in the Patent Application.

Figure 10c is a diagram illustrating an example data structure for LBA-SDRAM mapping table according to an embodiment of the present invention. If the SRAM is also used as a cache in the same manner as the SDRAM, then the LBA-SDRAM table also applies to data in the SRAM. The SDRAM/SRAM provides the fastest access to the host, hence all data written by the host are first buffered in the SDRAM/SRAM. Also, data read by the host are first read off the flash or hard drive to the SDRAM/SRAM. The LBA_SDRAM tables refer to the data structures maintained by the embedded processors in order to associate host data accesses to their locations in the SDRAM/SRAM cache. Each entry in the table such as 1036 associates a set of LBAs addressed by the host to information regarding their temporary location in the SDRAM/SRAM, and their original or eventual location in non-volatile storage, such as flash memory, a hard drive or equivalent. In general, the LBA_SDRAM tables include information such as: the location of the cached data blocks in the SDRAM/SRAM 1037, control information such as the caching state 1038 of the data blocks, etc. The location information allows the SDRAM DMA controller or SRAM controller to physically control the SDRAM devices and access the data. The caching state indicates how the cached data in the SDRAM differs compared to the version cached in the flash or stored in hard drives. The caching state 1038 include information such as: if that set of data is not yet allocated permanent storage, if they are fully or partly modified, if they are currently in transit and
should not be allowed access yet, etc. If the data is designated for storage in the flash or hard drives, then there is a corresponding entry in the LBA_Flash_HDD tables. Additional details of the optimized method for maintaining the LBA-SDRAM mapping table illustrated in Fig 10b are disclosed in the Patent Application.

Figure 11 is a diagram illustrating a power up initialization process flow applicable to the embodiments of the present invention as illustrated in figure 1, 2 and 3. This process pertains to the movement of the codes and the movement and update of the control structures and data. The initial code loaded to the embedded processor internal cache memory is stored in a non-volatile memory, such as FPROM 1101. An initial portion of the code executed by IO processor transfers the rest of the code in the FPROM to the SRAM 1102, which is the memory providing fastest-access to the IO processors and serves as a level-2 cache to the IO processor. Since the FPROM is a small capacity device, the rest of the code for the IO processor and other codes such as an OS kernel or applications optionally run by another embedded processor are stored in the flash. Portions of these codes are paged to the SRAM for execution 1103. The initial code loaded from the FPROM to the SRAM includes the routines for instructing the flash DMA controller to page the next set of routines to be executed 1104. Following the routines for paging code from the FPROM to the SRAM, the power up initialization also entails partitioning of the SDRAM into areas for caching data and areas for storing control structures. The flash DMA controller is instructed to fetch an initial set of control structures e.g. the control structure that holds the location information of the LBA_Flash_HDD tables in the flash 1105. The next step is to initialize the LBA_SDRAM tables to indicate the SDRAM cache area is empty 1106. After this initialization to the SDRAM, the system is ready to commence normal operation 1107. During normal operation, the IO processor services read/write requests from the host 1108 as well as manage the
different storage media (SRAM 1109, SDRAM 1110, flash 1111 and HDD 1112). The other processor(s) can run other applications and so the IO processor should perform the operations necessary to facilitate this 1113. As data is transferred between the different devices in the system, management of the storage media entails updates to the different control structures as well as periodic saving of such structures to the non-volatile storage media, such as flash memory and hard disks.

Figure 12 is a diagram illustrating a process flow of a block read command from a host computer system primarily applicable to the embodiments of the present invention as illustrated in figures 5, 7 and 9. The left side of the figure shows activities of the host system performing a block read operation on the hybrid storage device 1201. The right side of the figure shows activities within the hybrid storage device upon receiving a block read command from the host system 1202. The block IO transfer protocol (ATA or SCSI) allows the storage device to queue up the received commands 1203 and to respond with the requested data blocks within an extended period of time. When a number of commands have been queued, then the IO processor can proceed to start retrieving and processing the commands 1204.

The IO processor processes the commands by translating the IO address in the command to its corresponding physical location among the various storage media within the device 1205. The hybrid storage device reduces the response time by using the flash as intermediate cache between the hard disk(s) and the SRAM/SDRAM where the IO interface DMA controller retrieves data 1206. In one embodiment of the present invention, the DMA controller only accesses data from the hard disk(s) 1207 if the data requested is not cached in the flash memory. Because the instructions to the DMA controllers can be linked, the IO processor can build several such instructions in the background and link them. Using the instruction link, the DMA controllers can automatically fetch the next instruction from
memory and perform the instructed transfer without additional intervention from the IO processor. Once the amount of data transferred from nonvolatile storage (flash and hard disk) to volatile (SRAM/SDRAM) storage reaches a pre-determined threshold 1208, the IO Interface DMA controller is triggered to start transferring data blocks to the host system 1209. Status information of the block read command is sent after the hybrid storage device delivers the requested data block 1210.

Figure 13 is a diagram that illustrates a process flow of a block write command from a host computer system primarily applicable to the embodiments of the present invention as illustrated in figures 5, 7 and 9. The left side of the figure shows activities of the host system performing a block write operation on the hybrid storage device 1301. The right side of the figure shows activities within the hybrid storage device upon receiving a block write command from the host system 1302. The block IO transfer protocol (ATA or SCSI) allows the storage device to queue up the received commands 1303 and to indicate when it is ready to receive the data blocks from the host system. When a number of commands have been queued, then the IO processor can proceed to start retrieving and processing the commands 1304.

The IO processor processes the commands by translating the IO address in the command to its corresponding physical location among the various storage media within the device 1305. The hybrid storage device reduces the response time by using the flash as intermediate cache between the hard disk(s) and the SRAM/SDRAM where the IO interface DMA controller writes data. Because the instructions to the DMA controllers can be linked, the IO processor can build several such instructions in the background and link them. Using the instruction link, the DMA controllers can automatically fetch the next instruction from memory and perform the instructed transfer without additional intervention from the IO processor. If necessary, data must be flushed back from the volatile storage (SRAM/SDRAM).
to the flash. In this case the flash DMA controller is triggered to transfer data from the SRAM/SDRAM to the flash. If necessary, data must be flushed back from the volatile storage (SRAM/SDRAM) to the hard disk. In this case the IO Interface DMA controller connected to the hard disk is triggered to transfer data from the SRAM/SDRAM to the hard disk 1307. When the available space in the volatile storage (SRAM/SDRAM) buffer reaches a second pre-determined threshold 1308, the IO Interface DMA controller is triggered to continue receiving data blocks from the host system 1309. Status information for the block write command is sent after the hybrid storage system is able to write all the data 1310.

Figure 14 is a diagram illustrating a process flow of a random access byte read request received via the system bus applicable to an embodiment of the present invention as illustrated in figures 2 and 3. The left side of the figure shows activities of the requesting device connected to the system bus 1401. The right side of the figure shows activities within the hybrid storage device upon receiving a random access byte read request 1402.

Given the address of the requested read data 1403, the PCI-Express/PCI-X/PCI DMA controller 111 as illustrated in figure 1 can look up that address in the CAM 112. If the CAM returns a valid match for the address 1404, then the index of the entry also returned by the CAM corresponds to the index of the data block in the SRAM or SDRAM that contains the requested read data. The SRAM 105 and SDRAM 108 are also illustrated in figure 1. The PCI-Express/PCI-X/PCI DMA can translate the data block index to the SRAM or SDRAM address and continue to read the data 1405. However, if the CAM does not return a valid match, then it means none of the data blocks currently cached in the SRAM or SDRAM contain the requested read data. In this case, the PCI-Express/PCI-X/PCI DMA controller shall inform the IO processor 103 illustrated in figure 1 and give the address 1406. The IO processor then uses mapping tables and the procedure illustrated in figure 17 to locate the data and transfer data.
from either the flash or the hard disk to a free data block location in the SRAM or SDRAM 1407. When the transfer is complete, the IO processor writes the requested read data address to the CAM entry whose index corresponds to the data block index in the SRAM or SDRAM that now contains the requested read data 1408. If the host system did not abort the request 1409,

then the PCI-Express/PCI-X/PCI DMA controller can proceed to detecting the valid CAM match and eventually to reading the requested data from the SRAM/SDRAM. This data is sent back to the host system by the PCI-Express/PCI-X/PCI DMA controller 1410. The checking of the address match in the CAM is repeated until all of the data of the request is read 1411 after which the hybrid storage device will wait for the next request 1412. The checking of the address match in the CAM will not be performed however in cases where the host opted to abort the request 1413.

Figure 15 is a diagram illustrating a process flow of a random access byte write request received via the system bus applicable to an embodiment of the present invention as illustrated in figures 2 and 3. The left side of the figure shows activities of the requesting device connected to the system bus 1501. The right side of the figure shows activities within the hybrid storage device upon receiving a random access byte write request 1502.

Given the address of the write request 1503, the PCI-Express/PCI-X/PCI DMA controller 111 as illustrated in figure 1 can look up that address in the CAM 112. If the CAM returns a valid match for the address 1504, then the index of the entry also returned by the CAM corresponds to the index of the data block in the SRAM or SDRAM where the data can be written. The SRAM 105 and SDRAM 108 are also illustrated in figure 1. The PCI-Express/PCI-X/PCI DMA can accept the write the data then write to the SRAM or SDRAM 1510 as it can translate the data block index to the SRAM or SDRAM address 1505. However, if the CAM does not return a valid match, then it means none of the data blocks currently
cached in the SRAM or SDRAM contain a data location that can be written with the data. In this case, the PCI-Express/PCI-X/PCI DMA controller shall inform the IO processor 103 illustrated in figure 1 and give the write address 1506. The IO processor then uses the procedure illustrated in figure 18 to get a data block location in the SRAM or SDRAM where the data can be written 1507. The IO processor will write the write address to the CAM entry whose index corresponds to the data block index in the SRAM or SDRAM that can now contain the requested write data 1508. If the host system did not abort the request 1509, then the PCI-Express/PCI-X/PCI DMA controller can proceed to detecting the valid CAM match and eventually to accept the write data and write it to the SRAM or SDRAM. This data is received from the host system by the PCI-Express/PCI-X/PCI DMA controller 1510. The checking of the address match in the CAM is repeated until all of the data of the request is accepted 1511 after which the hybrid storage device will wait for the next request 1512. The checking of the address match in the CAM will not be performed however in cases where the host opted to abort the request 1513.

Figure 16 is a diagram illustrating a process flow of a DMA transfer according to an embodiment of the present invention. The left side of the figure shows activities of the IO processor upon determining that it needs to instruct a DMA controller to perform a DMA transfer 1601. The right side of the figure shows activities of a DMA controller upon being activated by the IO processor to perform a data transfer 1602. Because the DMA instructions can be linked, the IO processor can build several such instructions in the background and link them. These instructions are created and stored in the SRAM/SDRAM 1603. When there is at least one valid instruction stored, the IO processor can activate the DMA controllers to fetch and process the instruction 1606. If there are more data to transfer, and the previous instruction has not yet been fetch, then the IO processor can create and store a new instruction for the data
transfer 1604 and link the new instruction to the previous instruction by adding the location of
the new instruction to the previous instruction 1605. Whenever the DMA controller is
activated, it can check for the location of the instruction 1607. The DMA controller can
automatically fetch the instruction from memory 1608 and perform the instructed transfer
without additional intervention from the IO processor 1609. If the instruction contains a valid
link to a next instruction, then it can continue to repeat the process of fetching 1610 the
instruction and performing the instructed transfer 1611. This process flow is performed as part
of the response to read/write requests from the host system, or as part of management functions
for the different storage media or for any other purpose that involves DMA transfers.

Figures 17, 18 and 19 are diagrams of an example of a basic caching algorithm that can
be applied to the multi-tiered storage system. Both data and control information are cached in
the system. The SRAM, being the media that provides the fastest access for the IO processor,
is ideal for caching control structures used by the IO processor such as the different mapping
tables which are also stored in the SDRAM. The SRAM can also serve as a level-2 cache for
storing the code run by the processors. However, it may also serve as a data cache to
supplement the SDRAM. In figures 17, 18 and 19, only the SDRAM is mentioned but the
SRAM may be used as to cache data as well. The data caching scheme is implemented by the
IO processor code and can thus be programmed to be optimized for the application of the
system. In particular, the caching scheme illustrated in Figures 17, 18 and 19 shows the usage
of the SDRAM as a Level-1 data cache and the Flash as a Level-2 data cache.

Figure 17 is a high-level flow chart showing how the mapping tables are used and
updated in response to a read request from a host system. The figure shows the option wherein
data for a read request that is stored in the Flash is not to be cached and instead is temporarily
stored in the SRAM. When the IO processor is processing a read request or command, it
checks if requested LBA is associated with an entry in the L1 table or LBA-SDRAM table 1701. If this is true and there is a valid entry 1702, then it extracts the SDRAM location from the entry then proceeds with the read 1703. If there is no valid entry 1704 then it means the data is not presently cached in the SDRAM so the LBA-Flash table or L2 table will be checked to determine if the data is cached in the Flash 1705. If there is a valid entry, then it extracts the Flash location from the entry 1706. The IO processor has the option of caching the data in the SDRAM or L1 cache or if the data will be temporarily buffered in the SRAM. Data in the temporary SRAM buffers are considered valid only for the current read request and will be overwritten as needed. If it is determined that the data should be temporarily buffered in the SRAM, then the IO processor uses a temporary buffer location in the SRAM 1707 then it transfers the data from the Flash location to the said buffer location 1708. The data can then be retrieved to respond to the read request or command 1709. If it is determined that the data should be cached in the SDRAM, then the IO processor gets a free L1 table entry 1710. It then transfers the data from the Flash location to the SDRAM location extracted from the L1 table entry 1711. When the data has been cached in the L1 cache, then the data can be retrieved to respond to the read request or command 1712. The L1 table will then be updated to indicate that the entry now corresponds to the data copied from the Flash 1713. If there is no valid entry in the L2 table, then the data is also not cached in the Flash. The data must be retrieved from the HDD so the IO processor will check if the requested LBA range is associated with a valid entry in the LBA-HDD table 1714. If there is no valid entry then this means the given LBA is not valid or is out of the range of the LBAs provided by storage system and an error will be returned 1715. If there is a valid entry in the LBA-HDD table, then IO processor gets a free L1 table entry 1716. It then transfers the data from the HDD location extracted from the LBA-HDD table entry to the SDRAM location extracted from the L1 table entry 1717. When the
data has been cached in the L1 cache, then the data can be retrieved to respond to the read request or command 1718. The L1 table will then be updated to indicate that the entry now corresponds to the data copied from the HDD 1719. The data is cached in L1 or SDRAM but there is also an option to cache the data in the L2 cache or the Flash. If this option is taken, then the data will be copied from the SDRAM location to the Flash location 1720. The corresponding L2 table entry for that location should then be updated to indicate it data retrieved from the HDD is now cached in that Flash location 1721.

Figure 18 is a high-level flow chart showing how the mapping tables are used and updated in response to a write request from a host system. When the IO processor is processing a write request or command, it checks if requested LBA is associated with an entry in the L1 table or LBA-SDRAM table 1801. If this is true and there is a valid entry 1802, then it updates the entry 1803 to indicate that the data block is dirty as the write request or command will overwrite the data currently stored in the SDRAM 1804. If there is no valid entry in the L1 table 1805 then the IO processor checks if there is a free L1 table entry 1806. If there is no free table entry then a flush operation must be triggered 1807. The flush operation will force some data blocks in the SDRAM to be copied to non-volatile memory, sometimes referred to as "permanent storage." This approach enables the data location in the SDRAM to be replaceable by the data of the new write request, as it is assured that the old data is now stored in a non-volatile memory. This also means the entry of the flushed data block is now freed up and can be updated to correspond to the data block of the new write request or command 1808.

Figure 19 shows the procedure for flushing of data back to the HDD and caching of data in the Flash both of which can be triggered as a background process 1901 or during writes when a cache full occurs 1902. To minimize cache full during writes, a minimum count of entries in the L1 table is kept eligible for getting replaced so write requests can be immediately
accepted 1903. An entry in the L1 can be replaced if the corresponding data block is clean
meaning it has been flushed back to either the Flash of the HDD. In the illustrated scheme, all
data access requests from the connected host system are read from and written to the SDRAM
which serves as the Level-1 (L1) cache. If there is an L1 table entry scheduled to be flushed to
the HDD 1904, then the data referred to by the L1 table entry is transferred from the SDRAM
to the HDD 1905. After which, the LBA-HDD table entry 1906 and the L1 table entry 1907
must be updated to indicate that the entry is now clean or that it can be now replaced since the
cached data in SDRAM has been stored in a non-volatile memory, such as the HDD in this
example. There are also L1 entries scheduled to be flushed to the Flash 1908. The data
referred to by the L1 table entry is transferred from the SDRAM to the Flash 1909. After
which, the L2 table entry must be updated to indicate that the corresponding data block is now
dirty, as the data cached in the Flash does not anymore match the data stored in the HDD 1910.
The L1 table entry must be also updated to indicate that the entry is now clean 1911, or that it
can be now replaced since the cached data in SDRAM has been stored in non-volatile memory,
which in this example is in Flash. The Flash serves as a Level-2, also referred to as L2 herein,
cache which can be used to store copies of certain portions of data that are stored in the HDD.
The algorithm that decides which portions to store or cache in the L2 cache may differ. In one
embodiment of the present invention, one criterion that may be used to determine which data to
store in the L2 cache may include date that has been most recently accessed.

Foregoing described embodiments of the invention are provided as illustrations and
descriptions. They are not intended to limit the invention to precise form described. In
particular, it is contemplated that functional implementation of invention described herein may
be implemented equivalently in hardware, software, firmware, and/or other available
functional components or building blocks, and that the data storage system maybe distributed
comprising devices connected through a network, and that the network may be wired, wireless, or a combination of wired and wireless. Other variations and embodiments are possible in light of above teachings, and it is thus intended that the scope of invention not be limited by the various embodiments disclosed herein but rather by the following claims.
In the Claims

We claim:

1. A data storage system for storing and retrieving computer data using one or more logical address without intervention from the external host or client system, the data storage system comprising:
   - an IO processor for controlling data input or output of a data storage system;
   - a disk drive for storing one or more blocks of data, including data transferred from the host and control data for the IO processor;
   - a non-volatile memory means for storing a plurality of data blocks, the data blocks including data transferred from the host, a cached portion of data stored in the disk drive and control data for use by the IO processor; a LBA_Flash_HDD first table for storing at least one logical address mapped to at least one physical address of said non-volatile memory means and at least one physical address of said disk drive; and
   - wherein the LBA_Flash_HDD HDD table enables memory access to said non-volatile memory means without intervention from the host.

2. The data storage system of claim 1 wherein the LBA_Flash_HDD table is updated from time to time according to a data access behavior pattern of the host computer without the host system intervention.

3. The data storage system of claim 1 further comprising:
   - a FAST memory means for storing one or more blocks of data, the data including data transferred from the host, a cached portion of data stored in the non-volatile_memory means and control data for the IO processor means;
a LBA_FAST table wherein at least one of logical addresses is searched in the LBA_FAST table and if found, the logical address is mapped to at least one FAST memory address providing an association between the logical address and a physical location of data and enabling access to the data without intervention from the host.

4. The data storage system of claim 3 wherein the LBA_FAST table and the LBA_Flash_HDD table are updated from time to time according to a data access behavior pattern of the host without intervention from the host.

5. The data storage system of claim 4 wherein the FAST memory means includes:

   a SDRAM memory for storing one or more blocks of data, including data transferred from the host, a cached portion of data stored in the non-volatile memory means, control data for the IO processor, and the LBA_FAST table;

   SRAM memory for storing one or more blocks of data, including control data for the IO processor, and for caching the LBA_FAST table; and

   wherein, during operation, the LBA_Flash_HDD table is stored in the non-volatile memory means and cached in the FAST memory means, a first cached portion is stored in the SRAM memory and a second cached portion containing the remainder of the LBA_Flash_HDD table is stored in the SDRAM memory and a copy of the LBA_Flash_HDD table is stored in the disk drive.

6. The data storage system of claim 5 further comprising:

   a SDRAM DMA controller for transferring data to or from the SDRAM memory, the SDRAM DMA controller responsive to at least one DMA instruction;
a non-volatile_DMA controller for transferring data to and from the non-volatile memory means, the non-volatile DMA controller responsive to at least one DMA instruction;
the IO processor for preparing at least one DMA instruction, the IO processor using the LBA_FAST table and the LBA_Flash_HDD table for mapping the one or more logical addresses; and
wherein the control data stored in the FAST memory means further includes at least one DMA instruction.

7. The data storage system of claim 5, further comprising:
a host DMA controller means for transferring data to or from the host, computer the host DMA controller responsive to at least one or more DMA instructions and for interfacing the host with the data storage system;
CAM memory means for storing a byte address look up table wherein the host DMA controller means transfers data to and from the host computer without the IO processor means preparing the DMA instruction if upon the byte address look-up table contains a valid entry for the data requested by the host computer;
an IO DMA controller means for transferring data to and from the disk drive means in response to at least one or more DMA instructions;
wherein the host computer is coupled to the host DMA controller means for interfacing to the data storage system; and
wherein the disk drives means is coupled to the IO DMA controller means via an IO interface selected from the group consisting of IDE/ATA, serial ATA, USB, FIREWIRE, SCSI, FIBER CHANNEL, and ETHERNET.
8. The data storage system of claim 5, further comprising:
   a first IO DMA controller means for transferring data to and from the host computer in
   response to at least one or more DMA instructions; and
   a second IO DMA controller means for transferring data to and from the disk drive
   means in response to at least one or more DMA instructions;
   wherein the host computer is coupled to the first IO DMA controller means for
   interfacing to the data storage system; and
   wherein the disk drives means is coupled to the second IO DMA controller means.

9. The data storage system of claim 5 further comprising:
   an IO DMA controller means for transferring data to and from the host computer in
   response responsive to at least one or more DMA instructions; and
   an external bus interface DMA controller means for transferring data to and from the
   disk drive means in response responsive to at least one or more DMA instructions;
   wherein the host computer is coupled to the IO DMA controller means for interfacing
   to the data storage system; and
   wherein the disk drives means is coupled to the external bus interface DMA controller
   means.

10. A data structure for storing mapping information of a data storage system, the data storage
    system comprising disk drive means, flash memory means, SDRAM memory means, and
    SRAM memory means, the data structure comprising:
    a LBA_FLASH_HDD table comprising one or more logical address, one or more
    corresponding flash address, and one or more corresponding disk drive address; and
a LBA_SDRAM table comprising one or more logical address, one or more corresponding SRAM address, and one or more corresponding SDRAM address;

wherein a first portion of a working copy of the LBA_Flash_HDD table is stored in the SRAM memory means, a second portion of the working copy is stored in the SDRAM memory means; and

wherein the LBA_SDRAM table is stored in the SDRAM memory means, and a cached portion of the LBA_SDRAM table is stored in the SRAM memory means.

11. A data storage system for performing memory operations on a mass storage unit in response to a host request received from a host, the data storage system comprising:

a means for processing program code in response to the host request, said means including an IO processor;

a first non-volatile memory DMA controller electrically coupled to said IO processor;

a first non-volatile memory electrically coupled to said first non-volatile DMA controller and for storing a first table and selected data transferred from the host, said first table for storing a plurality of logical addresses respectively mapped to at least one physical memory address;

program code for mapping a first logical address to a physical address of a first data location in said first non-volatile memory; and

wherein, without requiring host intervention, said means for processing uses said first table when performing a memory operation on said first non-volatile memory.

12. The data storage system of claim 11, further including:
a mass storage DMA controller electrically coupled to said IO processor and for
electrically coupling to the mass storage unit; and

wherein said program code for further mapping said first logical address to a physical address of a second data location in the mass storage unit.

13. The data storage system of claim 12, further including:

a first volatile memory and a first volatile memory DMA controller electrically coupled to said first volatile memory and to said IO processor; and

wherein the mass storage unit includes at least one hard disk drive.

14. The data storage system of claim 13, wherein said first volatile memory for caching a portion of said selected data stored in said first non-volatile memory; and

a second table for storing a plurality of logical addresses that are each respectively mapped to at least one physical address, said plurality of logical addresses including a second logical address mapped to a physical address of a third data location in said first volatile memory; and

wherein said means for processing uses selected contents of said second table when performing a memory operation on said first volatile memory.

15. The data storage system of claim 12, further including:

a second volatile memory; and

wherein said first volatile memory for storing said second table, said second volatile memory for caching at least a portion of said second table; and said second volatile memory includes SRAM.
16. The data storage system of claim 12, further including program code for storing a copy of said first table in said mass storage unit in response to a selected event and said means for processing further includes a scratch pad buffer.

17. The data storage system of claim 11, wherein said means for processing uses selected contents of said first table when performing a memory operation on said mass storage unit.

18. The data storage system of claim 17, wherein said selected contents of said first table include said plurality of logical addresses respectively mapped to at least one physical address.

19. The data storage system of claim 12, further comprising:

   a host DMA controller for transferring data to or from the host in response to one or more DMA instructions;

   a CAM for storing a byte address look up table, and in response to the byte address look-up table containing a valid entry for data requested by the host, said host DMA controller transfers data to and from the host without the IO processor preparing the DMA instruction; and

   an IO storage DMA controller for transferring data to and from the mass storage unit in response to one or more DMA instructions and through an I/O interface.

20. The data storage system of claim 12 further comprising:

   a first IO means for transferring data to and from the host in response to at least one DMA instruction, said first IO means for interfacing with the host; and
a second IO means for transferring data to and from the mass storage unit in response to at least one DMA instruction, said second IO means for interfacing with the mass storage unit.

21. The data storage system of claim 20, wherein said first and second IO means respectively include a DMA controller.

22. The data storage system of claim 11, wherein said IO processor creates a link of DMA instructions in response to said host request; and wherein said first non-volatile memory DMA controller uses said link of DMA instructions to transfer information to or from said first non-volatile memory.

23. The data storage system of claim 12, further including a second non-volatile DMA controller electrically coupled to said IO processor and for coupling to a second non-volatile memory; and wherein said program code for further mapping said first logical address to a physical address of a second data location defined in said second non-volatile memory.

24. The data storage system of claim 12, wherein said first non-volatile memory includes a flash memory, and said second non-volatile memory includes a hard disk drive; and wherein said program code for further mapping said first logical address to a physical address of a second data location defined in said second non-volatile memory.

25. The data storage system of claim 12, further including a first volatile memory.
26. The data storage system of claim 25, further including a first volatile memory controller and wherein said first volatile memory includes at least one SRAM.

27. The data storage system of claim 26, further including:

   a bus;

   said first volatile memory controller electrically coupled to said IO processor through said bus; and

   wherein said first volatile memory includes at least one DRAM and is coupled to said bus.

28. The data storage system of claim 26, further including a read only memory.
Hybrid Storage Device as a Random-Access Slave Device attached to the System Bus

FIG. 2
Hybrid Storage Device as the Host System with a System Bus-Attached Slave Device

FIG. 3
Standalone Host System using a Hybrid Storage Device with Network Interface

FIG. 4
Hybrid Storage Device as a Block-Access Device attached to an IO Bus using an *Internal* IO Interface Controller

**FIG. 5**
Hybrid Storage Device as a Host System with a network interface and controlling block-access storage attached to an IO Bus using an External IO Interface Controller

FIG. 6
Hybrid Storage Device as a Block-Access Device
attached to an IO Bus using an External IO Interface Controller

FIG. 7
Hybrid Storage Device as the Host System with an IO Bus-Attached Slave Device and Network Interface

FIG. 8
Hybrid Storage Device as a Block-Access Hybrid Hard Disk

Host System
- IO Interface Controllers
  - IDE/ATA
  - SERIAL ATA
  - USB
  - FIREWIRE
  - SCSI
  - FIBER CHANNEL
  - ETHERNET

- HOST CPU
- MAIN MEMORY SDRAM
- ROM
- PCI-Express / PCI-X / PCI Bridge
- Other PC Peripherals

Hybrid Storage Controller
- IO Storage Interface DMA Controllers
  - IDE / ATA / SERIAL ATA
  - USB / FIREWIRE / SCSI / FIBER CHANNEL / ETHERNET

- External Bus Interface DMA Controller

- Hybrid Storage Device
  - Disk Control Electronics
  - Spindle motor and electromechanical assembly
  - Magnetic disk drive head and platters

FIG. 9
<table>
<thead>
<tr>
<th>LBA Set Idx</th>
<th>PBA</th>
<th>Flash Remap Table</th>
<th>HDD Remap Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x1</td>
<td></td>
<td></td>
<td>Entry for Host LBA set 1</td>
</tr>
<tr>
<td>0x2</td>
<td></td>
<td></td>
<td>Entry for Host LBA set 2</td>
</tr>
</tbody>
</table>

LBA-FLASH-HDD Table

**FIG. 10b**
<table>
<thead>
<tr>
<th>LBA Set Idx</th>
<th>Location</th>
<th>Control Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0</td>
<td>Entry for Host LBA set 0</td>
<td></td>
</tr>
<tr>
<td>0x1</td>
<td>Entry for Host LBA set 1</td>
<td></td>
</tr>
<tr>
<td>0x2</td>
<td>Entry for Host LBA set 2</td>
<td></td>
</tr>
</tbody>
</table>

**FIG. 10c**
Power-up

Load and run initial boot portion of code from non-volatile storage (FPROM or Flash)

Transfers rest of boot code from FPROM or Flash to SRAM

Run code in SRAM

Instruct Flash DMA controller to fetch rest of code from Flash to SRAM

Instruct Flash DMA controller to fetch LBA-Flash-HDD tables from Flash to SDRAM and SRAM

Initialize LBA-SDRAM tables to indicate SDRAM cache area is initially empty

Ready for normal operation

Service read/write requests from host system
Manage SRAM
Manage SDRAM cache
Manage flash memory
Manage HDD

Multi-tiered Caching Scheme implemented by IO Processor

Optionally, perform initializations to configure other processors to run applications over a conventional OS

FIG. 11
FIG. 12
Host System

Host System Sends Block Write Command using the IO Interface Controller (IDE/ATA, SATA, SCSI, Ethernet, USB, Firewire, Fibre Channel)

Command

Hybrid Storage Device receives Write Command through the IO Interface DMA controller and saves it to the preallocated command buffer

IO processor allocates buffers in volatile memory (SDRAM/ SRAM) and informs IO Interface DMA controller of allocated buffer

Command Queuing threshold met?

IO processor retrieves, checks, and interprets Commands

IO processor translates IO address (LBA) to SRAM/SDRAM addresses using mapping table and caching scheme (Fig18)

If data blocks should be flushed back to Flash, IO processor instructs Flash DMA Controller to transfer Data from SDRAM/ SRAM to Flash

If data blocks should be flushed back to HDD, IO processor instructs IO Interface DMA Controller to transfer Data from SDRAM/ SRAM to HDD

Data in SRAM/SDRAM is enough to start IO transfer?

Data ready for transfer

IO processor instructs Interface DMA Controller to start receiving data from SRAM/SDRAM to Host

IO Interface DMA Controller receives data and transfers it to allocated SRAM/SDRAM location

Data transfer complete for the command?

IO interface DMA controller sends completion/status information

Ready for Data

Data Transfers

15/21

FIG. 13
Host System \(\rightarrow\) 1401

PCI/PCI-X/PCI-Express Interface DMA Controller reads the internal block address index from the CAM using the system bus address.

- Valid CAM entry for system bus address?
  - Yes: 1405
    - The PCI-Express/PCI-X/PCI DMA Controller translates block index read from CAM to SRAM/SDRAM location and reads the Data from said SRAM/SDRAM location.
  - No: 1404

Hybrid Storage Device sends data byte(s) through the PCI-Express/PCI-X/PCI DMA controller.

More Data in request?
- Yes: 1409
- No: 1411

Idle until next request

Request aborted by Host system?
- Yes: 1413
- No: 1408

The PCI-Express/PCI-X/PCI DMA Controller informs of IO processor of read request with no corresponding data in the SRAM/SDRAM (uncached). It cannot accept data bytes until there is a valid CAM entry. The Host system could opt to abort request.

IO processor gets SRAM/SDRAM address for requested data blocks using mapping table and caching scheme (Fig 17).

Note: If data blocks should be read from HDD, IO processor instructs IO Interface DMA Controller to transfer Data from HDD to SDRAM/SRAM.

If data blocks should be read from Flash, IO processor instructs Flash DMA Controller to transfer Data from Flash to SDRAM/SDRAM.

IO Processor allocates SDRAM/SDRAM buffer for Read Data and writes a valid entry (system bus requested read address) to CAM after data transfer completes.
17/21

Host System

PCI/PCI-X/PCI-Express connected Host System writes to Hybrid Storage Byte-addressed location

1501

Data Write Request

Hybrid Storage Device

1502

IO processor implements caching scheme to determine distribution of data between SRAM/SDRAM, Flash Array and Hard disks

1503

The PCI-Express/PCI-X/PCI Interface DMA Controller reads the internal block address index from the CAM using the system bus address

1504

Valid CAM entry for system bus address?

No

1505

The PCI-Express/PCI-X/PCI DMA Controller translates block index read from CAM to SRAM/SDRAM location and writes the Data to said SRAM/SDRAM location

Yes

Hybrid Storage Device receives data byte(s) through the PCI-Express/PCI-X/PCI DMA controller

1506

IO processor gets SRAM/SDRAM addresses to contain data blocks using mapping table and caching scheme (Fig 18)

Note: If data blocks should be flushed back to HDD, IO processor instructs IO Interface DMA Controller to transfer Data from SRAM/SDRAM to HDD

1507

If data blocks should be flushed back to Flash, IO processor instructs Flash DMA Controller to transfer Data from SRAM/SDRAM to Flash

1508

IO Processor allocates SDRAM/SRAM buffer for Write Data, writes a valid entry (system bus requested write address) to CAM

1509

More Data in request?

Yes

1511

Request aborted by Host system?

No

1512

Idle until next request

No

1513

FIG. 15
Start - Service Read request/command

Check if requested data is associated with an entry in the L1 Table (LBA-SDRAM table)

Valid L1 table entry?

Yes → Do Read from SDRAM location

No → Valid L2 table entry?

Yes → Get location in Flash

No → Check if requested LBA range is associated with an entry in the L2 Table (LBA-Flash Remap table)

Valid L2 table entry?

Yes → Get location in table entry?

Yes → Update L1 cache?

Yes → Get free L1 table entry

No → Transfer data from HDD location to SDRAM/SRAM cache location

Do Read from SDRAM/SRAM cache location

Transfer data from Flash location to SDRAM/SRAM cache location

Update entry in L1 Table (LBA-SDRAM table)

LBA range eligible for caching in L2 cache?

Yes → Transfer data from SDRAM location to Flash location

Update entry in L2 Table (LBA-Flash table)

No → Return error

Get SRAM temporary Buffer location

Transfer data from Flash location to SRAM temporary Buffer

Do Read from SRAM temporary buffer

Get free L1 table entry

Update entry in L1 Table (LBA-SDRAM table)

Update entry in L2 Table (LBA-Flash table)

End
Start – Service Write request / command

Check if requested LBA range is associated with an entry in the L1 Table (LBA-SDRAM table)

Valid L1 table entry?

Yes

Update L1 table entry

Do Write to SDRAM

End

No

Free L1 table entry available?

Yes

Do Flush operation (Fig 19)

Get free L1 table entry

No

FIG. 18
Start: Activate as background process

Count of free L1 table entries less than minimum required?

Yes

Get L1 table entries scheduled to be flushed to HDD

Transfer data from SDRAM to HDD

Update LBA-HDD Table entries

Update L1 table entries

Get L1 table entries scheduled to be cached to Flash

Transfer data from SDRAM to Flash

Update L2 table entries

Update L1 table entries

end

FIG. 19