Network-on-SSD: A Scalable and High-Performance Communication Design Paradigm for SSDs

Arash Tavakkol*, Mohammad Arjomand* and Hamid Sarbazi-Azad*†
*HPCAN Lab, Computer Engineering Department, Sharif University of Technology, Tehran, Iran
†School of Computer Science, Institute for Research in Fundamental Sciences (IPM), Tehran, Iran
{tavakkol,arjomand}@ce.sharif.edu, azad@{sharif.edu, ipm.ir}

Abstract—In recent years, flash memory solid state disks (SSDs) have shown a great potential to change storage infrastructure because of its advantages of high speed and high throughput random access. This promising storage, however, greatly suffers from performance loss because of frequent "erase-before-write" and "garbage collection" operations. Thus, novel circuit-level, architectural, and algorithmic techniques are currently explored to address these limitations. In parallel with others, current study investigates replacing shared buses in multi-channel architecture of SSDs with an interconnection network to achieve scalable, high throughput, and reliable SSD storage systems. Roughly speaking, such a communication scheme provides superior parallelism that allows us to compensate the main part of the performance loss related to the aforementioned limitations through increasing data storage and retrieval processing throughput.

Index Terms—Flash memory, Solid state disk, Inter-package parallelism, Interconnection network.

1 Introduction

DUE to *high-latency* access to randomly-addressed data and *low-throughput* concurrent access to multiple data, widely-used rotating Hard Disk Drives (HDD) are known to be performance- and bandwidth-limited mediums. Flash-based Solid State Disks (SSD), on the other hand, are becoming the mainstream *high-performance* and *energy-efficient* storage devices because of their fast random access, superior throughput, as well as low power features. Leveraging such a promising medium, researchers have made extensive studies to exploit this technology in storage systems and thus proposed solutions for performance optimization. Most of these studies have focused on either improving SSD technology limitations (e.g. slow random write access), lowering performance overhead of *erase-before-write* and *garbage collection* processes, or enabling access load balancing by means of circuit-level, architectural, or algorithmic techniques.

As random writes to flash memory are slow and maximum parallelism within a package is limited to concurrent access of including planes, one flash package can only provide a limited bandwidth of 32-40MB/s [1]. Therefore, SSDs usually utilize a multi-channel multi-way bus structure to conduct multiple data accesses in parallel while providing maximum stripping likelihood. This enhances aggregate bandwidth of a set of flash packages and improves back-end reliability. Although increasing the number of bus channels and hence the level of parallelism results in increased performance and bandwidth, there has to be tradeoffs between flash controller complexity, design cost, and maximum achievable throughput. These tradeoffs are even more challenging as the need for better performance remains due to the stiff increase of processor parallelism in many-core systems.

In this paper, we suggest replacing shared multi-channel bus wiring with an interconnection network. Using a network can result in higher performance, larger aggregate bandwidth, more scalability, and better reliability. In fact, interconnection network structures the global wires so that their electrical properties are optimized and well-controlled. These controlled electrical parameters, finally, result in reduced propagation latency and increased aggregate bandwidth [3]. Moreover, sharing the communication resources between many read/write requests makes more efficient use of the resources: when one set of flash packages are idle, others continue to make use of the network resources.

In a nutshell, networks are generally preferable to shared buses because they have higher bandwidth and support multiple concurrent communications. Note that some of our motivations for inter-flash package network stem from inter-chip networks in general purpose multi-processors and intra-chip networks (Network-on-Chip) in chip multiprocessors.

2 The SSD Structure

Figure 1 illustrates internal structure of an SSD that uses some microchips of non-volatile memories to retain data. As of 2012, most SSDs use *NAND flash memories* to provide a high dense medium with random read and write access capabilities. Each NAND flash package consists of multiple dies and each die corresponds to multiple planes containing the actual physical memory pages inside (e.g. 2, 4, or 8KB). Though each die can perform read, write, or erase operation independent of the others, all planes within a die can only carry out same (or at most two) command(s) at a time. To support parallelism inside a die, each plane includes an embedded data register to hold data when a read or write request is issued. Both read and write operations are processed in page units. Nevertheless, flash memory pages must be erased before any write which incurs a latency of milliseconds. To hide this large stall time, the unit of an erase operation in NAND flash memories is generally in tens of consecutive pages, forming an *erase block*. To further

Manuscript submitted: 07-Feb-2012. Manuscript accepted: 28-Feb-2012. Final manuscript received: 04-Mar-2012.

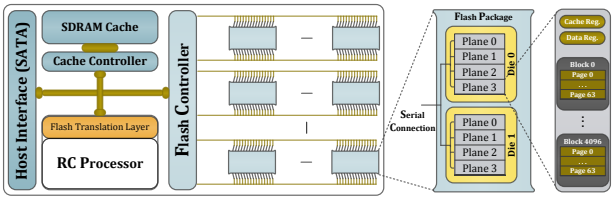

Fig. 1. A shared bus-based SSD internal structure.

reduce frequent erases, current flash memories use a simple invalidation mechanism along with out-of-place update [1].

In order to emulate a block-device interface provided in conventional HDDs, an SSD contains a special software layer, namely Flash Translation Layer (FTL). FTL has two main roles: the *address mapping* for translating logical addresses to physical addresses and *garbage collection* for reclaiming invalid pages. Moreover, FTL is responsible for prolonging the lifetime of NAND memories through a *wear-leveling* process. In more details, as the number of reliable erase cycles onto each flash page is limited to about 10^5 times [6], this process tries to reduce the erase count and make all blocks within an SSD to wear out evenly. To realize FTL processes, an embedded RC processor augmented with an SDRAM buffer (responsible for keeping block mapping information and acting as a write buffer) is utilized.

Considering a single flash package, limited support of parallel access to planes of a single die and high-latency random write limit the maximum provided bandwidth to 32-40MB/s [1]. In addition, *garbage collection* and *erase-before-write* operations require repetitive high-latency write and erase operations which incur large latencies. To overcome these shortcomings, a multi-channel bus structure is used to connect flash packages to the embedded controller. In this structure, a group of flash packages share a single serial bus where a controller dynamically selects the target of each command while a shared data path connects flash packages to the controller. This configuration increases capacity without requiring more pins, but it does not increase the bandwidth.

By enhancing parallelism through increased number of parallel channels, the SSD performance has been improved rapidly over the last few years. So that cutting-edge SSDs now exhibit sustained read/write throughput of up to 250MB/s. However, the strong demand for storage devices with better performance and bandwidth (1GB/s as projected in [4]) is increasingly developed.

3 NoSSD: Network-on-SSD

As mentioned before, the solutions for communication structure between flash packages and the embedded controller have generally been characterized by design of multi-channel multi-way buses. Although channels in this architecture build on well-understood on-board routing concepts with minimum complexity, enhancing parallelism in terms of increased number of channels has some advantages and some disadvantages. On one hand, high concurrency would generally improve resource utilization and increase throughput. On the other hand, addressing the channel access arbitration problem in the controller is not trivial and its latency and cost overhead may not be tolerable. For instance, a fully-interconnected structure (one outgoing channel per flash package) is optimal in terms of bandwidth, latency and power usage. However, the controller complexity in this structure linearly increases with the number of flash packages. Thus, there always has to be tradeoffs between the controller complexity, cost, and the maximum throughput. On the contrary, attaching more flash packages to a shared channel in a bus-based communication structure largely degrades the overall SSD performance and power efficiency. Indeed, this inefficiency is a direct effect of increased access contention and capacitive load of the attached units.

Lack of pipelining support is the other shortcoming of shared channels. More accurately, when depth of the service queue in the controller increases over the number of physical channels, a single channel has to be shared by more than one job which must be serialized [2]. In short, any change in communication structure, supporting pipeline data transmission, would substantially affect the aforementioned performance and power loss.

For maximum flexibility and scalability, it is generally accepted to move towards a shared, segmented network communication structure. This notion translated into a data-routing network consisting of communication links and routing nodes, namely *Network-on-SSD* (NoSSD). In contrast to the shared channel communication scheme, such a distributed medium scales well with package count and internal parallelism. Additional advantages include increased bandwidth as well as improved throughput and reliability by exploiting the provided parallelism in NoSSD.

3.1 NoSSD Design Concepts

Figure 2 illustrates topological aspects of a sample NoSSD structured as a 4-by-4 grid that is connected to the flash controller from one side (one outgoing channel per row for the controller). The NoSSD, in a simplified perspective, contains *routing nodes* to route data according to chosen routing protocol and *inter-router* links connecting the nodes. Each routing node has to be logically embedded within the flash access circuitry, so that routing the inter-node links onto SSD board shows more regularity.

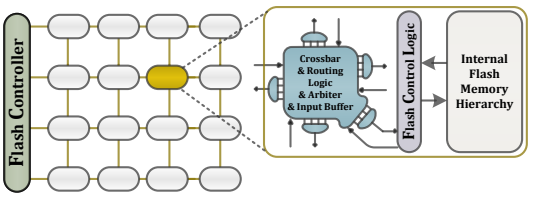

Fig. 2. Logical view of a NoSSD with 4-by-4 grid topology.

Details of the input-buffered router element alongside the flash memory logic are shown in Figure 2. The router connects to the local flash and adjacent units through bidirectional channels. A local port is used to connect flash logic to the network substrate while other fours provide connections to the neighbors in grid topology. This rich connectivity, however, is negatively limited by package pin count. Alleviating this constraint, we assume that data and control pins of a flash package are now multiplexed, forming a wide access port (Section 3.2). This port is then uniformly separated into parts each connected to an adjacent package.

Considering commodity design of buffer storage, routing algorithm, and arbiters, router logic is simple with few hundred gates per flash package. Among router elements, the area of a router is dominated by buffer storage. If we assume one-flit of buffering for each input channel (with 8 bit width as shown in Figure 3), the total buffering requirement becomes about 40 bits per router. Our estimation shows the routing logic, arbiters, and buffer storage will occupy an area less than 1% of a dense flash chip. In addition to this area, we assume same inter-package wiring layout as shared-channel SSDs use which makes the wiring effect on design complexity to be substantially less than router design.

3.2 Messaging Protocol in NoSSD

Using the concepts from interconnection networks, an NoSSD can use wormhole packetization model [3] in which control information and raw data are all integrated into packets. In more details, each packet carries read/write data, flash access commands, and network address information. The packet format, shown in Figure 3, consists of header flow control units (flits) followed by the payload flits. Header carries destination address, request/response information, and packet size while payload flits include raw data. An additional 1-bit head identifies the type of a flit that can be head or payload. A routing logic decides on the output port for a message when its header reaches the buffer head of an input port. When the route is determined, the message waits until the next hop buffer is available and preempts an output channel after arbitration. A crossbar switch uses arbitration signals to provide synchronous connections between any pair of input and output lines. Finally, a flit physically transmits over a channel. When a message reaches its destination, the local port is allocated and fed into flash control logic flit by flit.

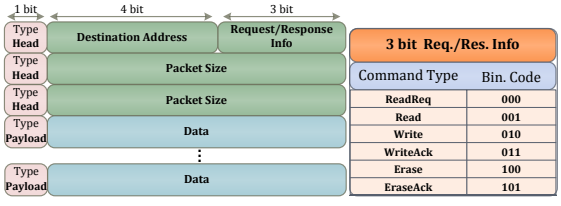

Fig. 3. Packet format for 4×4 NoSSD messaging protocol.

To access a page for read operation, a read command is encapsulated into a packet (i.e. *ReadReq* packet) and is issued from the controller toward the target flash package. The response would be returned back by the destination through a *Read* packet which includes both data of the read page and its meta-data (e.g. CRC and wear-leveling information). For write operation, on the other hand, the controller initially issues a packet (i.e. *Write* packet) containing both write command and data with target package as destination. In reply and after a successful write, flash package forms a *WriteAck* packet with controller as destination and meta-data as payload which will be then injected into the network for routing. The same mechanism is applied for erase operation through *Erase* and *EraseAck* packets.

3.3 Routing Mechanism in NoSSD

Similar to conventional SSDs, NoSSD is designed as a modular architecture but some components are added to obtain a fully-pipelined inter-package communication structure with specific characteristics (e.g. messaging protocol). Compared to interconnection networks, NoSSD has some common characteristics and specific features as follows.

NoSSD uses synchronous pipelined router architecture in order to be compliant with available network and flash memory standards. The data transmission between routers, on the other hand, is asynchronous. To reduce data transfer latency in the router and to increase the communication link bandwidth accordingly, the router can be designed to support one routing logic in each incoming port. Thus, NoSSD can forward maximum number of flits simultaneously from different incoming-outgoing data switching pairs.

NoSSD guarantees lossless and in-order packet delivery by means of utilizing a link-level flow control mechanism between routers to avoid data overflow. Therefore, the size of the FIFO buffers in the routers can be freely determined to keep a tradeoff between the maximum tolerable load and area consumption.

To keep network overhead at an acceptable level, routing logic has to be minimal (one with hop count equal to Manhattan distance, e.g. turn model routings [3]). Regarding minimal routing, different policies for packet injection can be taken at the flash controller. To reduce network hop count, a trivial choice is to deterministically inject a packet to the network through the channel connected to the same row that the target flash package is attached to (known as *nearest channel*). Though such a routing configuration simplifies controller design, it completely bypasses using vertical inter-package links and reduces network utilization as a result. More precisely, this scheme models NoSSD as a pipelined shared bus structure.

To further increase the network efficiency in cost of increased average hop count per packet, the controller may decide on the outgoing channel considering load balancing. A simple scheme is to allocate outgoing channels to packets using a *round robin* scheme, if *nearest channel* cannot be assigned to. Though simple, this scheme never guarantees load balancing especially when destinations are randomly chosen by FTL. An alternative is *intelligent routing* in which the allocated channel is the one with both under-threshold load and minimum hop count to the destination. Surely, *intelligent routing* requires a load characterization scheme at controller outgoing channels (e.g. counters) which, in turn, increases the controller complexity.

3.4 SSD Design Opportunities using NoSSD

Replacing shared channels with NoSSD provides some advantages and new opportunities.

To facilitate data movements during *garbage collection* and *erase-before-write* operations, most SSDs model page reclaiming as a sequence of read, write, and erase requests handled by the controller. Though simple and straightforward, this mechanism increases load of the controller and channels. Using NoSSD's inter-package links (vertical or horizontal) and making slight changes in the messaging protocol, we can mitigate this inefficiency and improve network utilization. As a solution, the controller can issue a *Clean* packet (using *reserved* commands) toward the reclaiming source package (*RS*). This packet must include addresses of the valid pages and the reclaiming destination package (*RD*). In response, RS augments the *Clean* packet with valid pages'

data, and then forwards it to RD. Finally, RD writes received data pages to the specified free positions and issues a *CleanAck* packet to notify the controller about reclaiming task accomplishment.

The NoSSD design in Figure 2 uses a grid topology with the controller positioned adjacent to one side. This simple configuration imposes variations on response time and throughput, especially when request size is small (such as TPCC and TPCE workloads [5]). In fact, for a NoSSD with p flash packages and c controller channels, there are $\binom{p}{c}$ choices to connect the controller to network nodes, each with different latency and throughput profile. To minimize the negative effects of the placement and sensitivity to the request mapping strategy, an optimization approach may be used. This way, FTL design complexity is reduced with more concern on wear-leveling and less stress on load balancing.

As mentioned before, the high latency of *garbage collection* operation can be tolerated using *Clean* and *CleanAck* packets. However, once a *Clean* request is issued toward a flash package, all subsequent read/write requests with the same target have to wait till the end of reclaiming. This increases the effective access latency. NoSSD, as a novel communication paradigm, may address such problems by prioritizing packets of certain classes over others within routers' arbitration units. In fact, the main goal is to speed-up accesses to the flash storage, so that less stall time is seen when crossing router input buffers. To this end, the router micro-architecture could be reconfigured in routing and arbitration logic to provide a certain class of prioritization using the command type encapsulated in early flits of the header.

4 Evaluation Results

We use Xmulator interconnection simulator [8] and Microsoft's SSD model based on DiskSim 4.0 [7] for NoSSD evaluation. The specification of the modeled flash memory [6] along with the parameters of baseline multi-channel SSD and NoSSD system is given in Table 1. Xmulator is a discrete-event trace-driven simulator, so we can evaluate NoSSD efficiency in terms of *response time*, *bandwidth*, and *throughput* metrics for several billions of transactions. To provide Xmulator with traces of read/write and erase/reclaiming requests within a SSD, we use extended DiskSim SSD model running real workloads. The considered workloads include IOZone, Postmark, TPCE, and TPCC applications.

TABLE 1
Parameters of the evaluation platform.

NAND Flash (Samsung Elec.)	Page size = $4KB$, Read latency = $25\mu s$, Write latency = $200\mu s$, Block erase latency = $1.5ms$, Flash package size = $4GB$ (IOZone, Postmark), $8GB$ (TPCC, TPCE).
Baseline SSD	Bandwidth = 16 bits, Propagation latency = $25ns$, Size = 4 channels × 4 ways (IOZone, Postmark), 4 channels × 8 ways (TPCC, TPCE).
NoSSD	Bandwidth = 8 bits, Inter router propagation latency = $12.5ns$, Input buffer size = 1 flit, Flit size = 8 bits, Network size = 4 × 4 grid (IOZone, Postmark), 4 × 8 grid (TPCC, TPCE).

The proposed NoSSD architecture is compared to shared-channel structure in terms of *average response time*, *throughput*, and *bandwidth*. Table 2 exhibits comparative analysis for 4×4 (in IOZone and Postmark) and 4×8 (in TPCC and TPCE) SSD structures for studied workloads. As can be seen, NoSSD makes slight improvement in terms of response time (about 2%) for TPCC and TPCE applications, where the average request size is small (typically, 2 pages of 4KB size [5]). However, it improves the response time up to 10% in IOZone and Postmark file-system benchmarks where the requests are considerably large (16 to 64 pages). On the other hand, NoSSD greatly improves the maximum achievable bandwidth (in MB) and throughput (in Kilo IOps), especially when running transactional applications such as TPCC. As depicted in Table 2, under TPCC and TPCE, sustained bandwidth and throughput improvements for NoSSD are up to 40%. Nevertheless, the improvement is much smaller (about 10%) for IOZone and Postmark applications.

TABLE 2
Comparative analysis of NoSSD and multi-channel SSD.

Workload	Response Time (ms)		Throughput (Kilo IOps)		Bandwidth (MB)	
	Bus	NoSSD	Bus	NoSSD	Bus	NoSSD
IOZone	2.496	2.156	0.780	0.834	229.8	242.9
Postmark	1.810	1.651	0.256	0.272	280.7	301.0
TPCC	0.138	0.136	44.8	64.5	355.8	511.5
TPCE	0.058	0.057	90.1	110.4	710.8	872.4

In summary, NoSSD can be a cause of bandwidth enhancement for OLTP applications where request size is small and the parallelism provided in NoSSD increases the maximum tolerable communication load. On the other hand, even though latency improvement per single message is not large in NoSSD, the aggregate latency improvement of a bunch of messages forming a request of a filesystem benchmarks is extremely high while it keeps SSD's response time low.

5 Conclusion

The performance of future large SSDs will be limited because of repetitive contentions on multi-channel shared buses connecting flash packages to the controller. To address this limitation, we have leveraged inter-package networks for low-latency high-throughput access to data pages. We proposed NoSSD, which efficiently provides a pipelined multi-router access to flash storage. The results revealed higher performance, larger bandwidth, more scalability, and better reliability with respect to conventional SSD structures.

References

[1] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance," Proc. USENIX'08, pp. 57–70, 2008.

[2] F. Chen, R. Lee, and X. Zhang, "Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing," Proc. HPCA'11, pp. 266–277, 2011.

[3] W.J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufman, 2004.

[4] R.F. Freitas and W.W. Wilcke, "Storage-class memory: The next storage system technology," IBM Journal of Research and Development, vol. 52, no. 4.5, pp. 439–447, 2008.

[5] S. Kavalanekar, B. Worthington, Q. Zhang, and V. Sharda, "Characterization of storage workload traces from production Windows Servers," Proc. IISCW'08, pp. 119–128, 2008.

[6] Samsung Electronics, K9XXG08XXM series, 2007.

[7] The disksim simulation environment. Ver. 4.0. Retrieved in September 2011 from: http://www.pdl.cmu.edu/DiskSim/

[8] Xmulator simulator. Ver. 6.0. http://www.xmulator.com/