Design for Scalability in Enterprise 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

Solid State Drives (SSDs) have recently emerged as a high speed random access alternative to classical magnetic disks. To date, SSD designs have been largely based on multi-channel bus architecture that confronts serious scalability problems in high-end enterprise SSDs with dozens of flash memory chips and a gigabyte host interface. This forces the community to rapidly change the bus-based inter-flash standards to respond to ever increasing application demands. In this paper, we first give a deep look at how different flash parameters and SSD internal designs affect the actual performance and scalability of the conventional architecture. Our experiments show that SSD performance improvement through either enhancing intra-chip parallelism or increasing the number of flash units is limited by frequent contentions occurred on the shared channels. Our discussion will be followed up by presenting and evaluating a network-based protocol adopted for flash communications in SSDs that addresses design constraints of the multi-channel bus architecture. This protocol leverages the properties of interconnection networks to attain a high performance SSD. Further, we will show and discuss that using this communication paradigm not only helps to obtain better SSD backend latency and throughput, but also to lower the variance of response time compared to the conventional designs. In addition, greater number of flash chips can be added with much less concerns on board-level signal integrity challenges including channels' maximum capacitive load, output drivers' slew rate, and impedance control.

Categories and Subject Descriptors

B.3.2 [**Memory Structures**]: Design Styles—*mass storage*; D.4.2 [**Operating Systems**]: Storage Management—*secondary storage*; B.4.3 [**Input/Output and Data Communications**]: Interconnections—*topology*

PACT'14, August 24–27, 2014, Edmonton, AB, Canada.

http://dx.doi.org/10.1145/2628071.2628098 .

Keywords

NAND flash memory; I/O Interface; Solid State Drive; Interconnection Network

1. INTRODUCTION

Enterprise applications are among the most costly workloads today in terms of IT infrastructure and resources, due to the complexities involved in running these systems and their huge storage requirements. An enterprise storage must meet capacity, performance, and reliability requirements, while minimizing the cost. Therefore, today's storage systems are steadily marching towards using Solid State Drives (SSDs), mainly because of their higher performance and lower power consumption compared to magnetic disks.

To enhance the overall performance and capacity, SSDs employ a multi-chip architecture where tens of flash memory chips are used. For generic SSDs, the overall bandwidth is usually bounded by the host interface, e.g., SCSI, PATA and SATA. Thus, multiple flash chips in the SSD usually share channels or buses to connect to the controller where channels are arranged in a way that their collective bandwidth matches the host interface bandwidth [8, 30, 48]. High-end SSDs in enterprise servers, on the other side, have a gigabyte host interface [48]. A straightforward approach to gain this bandwidth in the backend is to design for maximum parallelism by using dedicated channels for each flash chip [31, 44]. This increases the complexity of flash controller and needs to utilize hardware accelerators [26] or hierarchal controllers [26] for high performance design. Hence, current enterprise SSDs tend to use more channels (yet relatively moderate values), e.g., 8 to 32 channels in PMC NVMe controllers [41, 42, 43], 8 channels in Marvell 88NV9145 family [35] and SandForce SF-2000 family [45].

In this paper, we focus on scalability analysis of the enterprise SSDs. Scalability in enterprise systems is usually achieved by adding more flash memory chips to a channel and render multiple chips busy by exploiting striping allocation [34, 48]. Despite its advantages, the scalability of this design is in jeopardy: although the bus concept is well understood and is easy to employ, it quickly becomes a communication bottleneck in a heavily-loaded channel. Therefore, electrical properties like slew rate and duty cycle distortion does not allow attaching more than a few chips to a channel, for example ONFI 3.2 allows to attach up to 8 flash units working at at maximum bandwidth of 533 MB/s [7]. As more flash chips are attached to the channel, the power usage and latency per communication event grow due to the larger collective capacitive load. Thus, scalability be-

comes an important concern in today's SSDs and flash standardization work groups are steadily seeking for solutions to enhance the properties of bus-based communication for more and more available bandwidth. This way, from 2008, we have seen successive introduction of the communication standards to enhance 50 MB/s bandwidth in conventional SDR to 533 MB/s in NV-DDR2 or Toggle DDR [5, 7]. This transfer rate cannot be achieved easily and designers must follow specific rules to keep good behavior in signal integrity such as On-Die Termination (ODT), differential clocks, and short channel wires [20, 49].

To cope with this issue, Tavakkol *et al.* [50] suggested to leverage a network-based controller-to-flash communication structure. Their proposal, named as Network-on-SSD (NoSSD), gives higher performance, larger bandwidth, more scalability, and better reliability with least concern about signal distribution problems. In other words, NoSSD is an interconnection network between flashes and the controller that pipelines the global wires, so that their electrical properties are improved and well controlled. These controlled electrical parameters, finally, enable the use of signaling circuits that can largely reduce communication propagation latency and increase the working frequency with much less concerns on the signal integrity. More importantly, sharing communication resources among many I/O requests makes more efficient use of the resources: when one set of flash memory chips are idle, others can continue using network resources. Unfortunately, Tavakkol's study suffers form lack of key design considerations, especially those related to the implementation details of network routers and a realistic messaging protocol for a practical model of NoSSD. In this paper and after our detailed scalability analysis, we will go over the proper choices for NoSSD routers' configuration along with details of the messaging protocol to conduct flash command execution using underlying network infrastructure.

To summarize, this paper has two novel contributions: 1) we extensively analyze the opportunities to scale SSD storage subsystem by examining effects of different flash parameters/structures and multi-channel architecture of SSDs. To the best of our knowledge, such a study is not accomplished in the past and our results will highlight the inefficiency sources of SSD scalability. 2) The proper configuration of the NoSSD routers is described based on board-level inter-flash communication requirements. Different NoSSD design rules, including packet prioritization policy, packet injection policy for controller, and controller to network communication placement are investigated to help achieving higher performance results. Finally, a messaging protocol is proposed to efficiently conduct NAND flash command execution through NoSSD. We hope that our findings help the academic and industrial communities to revise inter-flash communications when thinking about high performance enterprise SSDs.

2. BACKGROUND

In this section, we describe the internal architecture of a standard enterprise SSD and summarize the structural characteristics of NAND flash as storage medium.

2.1 The NAND Flash: Organization and Operation

In a NAND flash, the unit of read and program (write) operations is one page of data but latency asymmetry is observed when a page is read or programmed; we can read any page in a NAND flash, but we must perform an erase operation before programming data to a page. The erase operation is performed at the granularity of a block consisting multiple adjacent pages. Regarding bit density of a cell, single level cell (SLC) and multi level cell (MLC) are two types of NAND flash. The capacity of MLC flash doubles that of SLC type, but SLC flash enjoys lower operational latency and higher endurance.

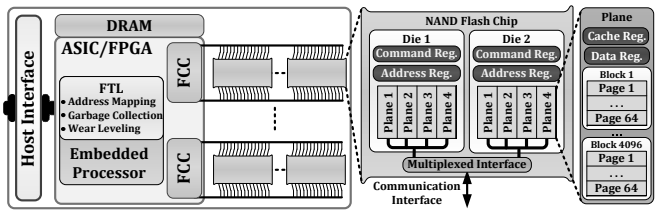


Figure 1: The internal architecture of enterprise SSDs and the structure of NAND flash memory chips.

Figure 1 shows the general organization of a flash memory chip. Within a chip, one or more dies share a single multiplexed interface. Each die can be selected individually and execute a command independent of others, which improves the overall throughput. Using a shared interface by all dies leads to a reduction in I/O bus complexity, but increases contention between requests. A die is typically composed of two or more planes, the smallest units to serve an I/O request. In fact, each plane has a separate wordline for accessing the flash memory pages, which results in an important consequence that multiple requests can be served simultaneously in different planes. However, to this end, the requests must adhere to the rules of multi-plane commands: the pages executing a multi-plane read/program operation must have the same die, block and page addresses; and the blocks executing a multi-plane erase operation must have the same die and block addresses. In addition to multi-plane commands, most of NAND flash memory chips provide other types of advanced commands to support die interleaving and copy-back execution. Interleaved commands provide parallelism within a chip, and copy-back commands enable a page to be copied within a plane without utilizing communication channels.

2.2 Enterprise SSD Internals

Our discussion is based on the architecture of standard enterprise SSDs, widely used in similar studies [17, 18, 38, 44, 48]. Variations of this architecture have been adopted in modern high-end enterprise SSDs, e.g., Fusion-io ioDrive2 Duo [18] and Intel910 series [26]. In enterprise SSDs, PCI Express is usually used as the host interface, where links provide a total bandwidth of 3.2-4 GB/s to exploit tens of flash chips in parallel [17, 48].

The high-level view of a multi-channel enterprise SSD is shown in Figure 1. The SSD uses a FPGA or ASIC between the PCI Express interface and flash channels to implement flash translation layer (FTL) and flash channel controllers (FCCs). FCCs are independently integrated into the FPGA/ASIC for each channel to maximize parallelism. Each FCC consists of a control logic, two independent buffers for read and write, and a NAND interface. There are usually tens of flash memory chips on the board and the controller is equipped with a large amount of RAM to keep FTL mapping

information or behave as a cache buffer. FTL is responsible for following tasks:

Address mapping. To emulate disk functionalities, FTL translates the host logical page addresses to internal physical page addresses through an address mapping policy.

Garbage collection (GC). The erase-before-write property and block-level erasure hinder in-place update of the page data, and hence the SSD controller uses a page invalidation mechanism in conjunction with an out-of-place update policy. This policy necessitates erase operation when device runs out of clean pages. A process called garbage collector is then triggered which selects a candidate block and rewrites its valid page into a free block and finally erases it. The garbage collection is very costly due to the required program/erase operations. To remedy this problem, FTL tries to keep user-visible capacity to a fraction of total physical pages, called over-provisioning, below which garbage collection is performed aggressively. The over-provisioning is typically expressed in terms of *spare factor* which is the ratio of added space to the total storage space [54]. Beside this mechanism, modern FTLs may trigger garbage collection process sooner, when SSD is idle and the number of free pages in a flash chip is low (known as background GC).

Wear leveling. Wear-out is one of the most oft-cited concerns about flash memory which causes physical damages to the cells and is not reversible. The datasheet of a flash chip reports a number of 1–100K erase cycles a block can undergo before it becomes unreliable. To this end, FTLs usually use caching, rescheduling, and uniformly distributing the writes to mitigate quick wear-out [11, 12].

3. EXPERIMENTAL METHODOLOGY

We perform discrete-event trace-driven simulation of enterprise SSDs with different communication models using SSDsim [24] augmented with Xmulator [4] interconnection network simulator. Although there are few famous SSD simulators, such as Microsoft SSD model for DiskSim [9], none of them provide a reasonable implementation of the controller-to-flash-chips communication model. However, SSDSim precisely implements ONFI signaling model and its accuracy has been verified using hardware prototyping [24]. Figure 2 depicts a logical view of our simulation methodology. SSDsim gets disk traces as the input workload and is capable of simulating all components shown in Figure 1, including FTL functionsalities and buffer management algorithms at FCC. Once a request is scheduled at a FCC, Xmulator is invoked. Xmulator is a parameterized interconnection network simulator that reports network latency and throughput estimations. Finally, the estimated communication latency and operation latency are sent back to SSDsim for future scheduling.

Enterprise SSD System. We model 1–2 TB enterprise SSDs detailed in Table 1, each with a spare factor of 30% [1]. As a high-end storage system, the configurations considered here have 8 or 4 independent channels each supporting 8 or 16 flash chips, respectively.

Controller Model. The simulated SSD controller runs a fully dynamic page level FTL [19], where a logical page address is not translated to a pre-determined plane by default.

[1]In lifetime-critical enterprise applications, higher spare factor limits are desired. For instance, Intel 910 series has a raw capacity of 1792 GB and user visible capacity of 800 GB (55% spare factor).

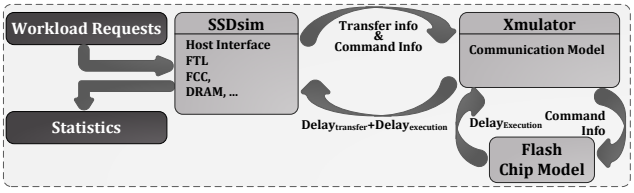


Figure 2: Simulation methodology.

Table 1: Main characteristics of simulated SSDs.

Multi channel SSD
8 Channels, 8 Flash Chips per Channel, Channel Width = 8 bit, NAND Interface Transfer Delay = 5 ns (ONFI NV-DDR), Page Allocation Strategy = Dynamic, Spare Factor = 30%
NAND flash (Micron [37])
Page Size = 4 kB, Metadata Size = 224 B, Block Size = 128 pages, 2 Planes per Die, 8 Dies per Flash Chip, Flash Chip Capacity = 32 GB, Read Latency = 35 µs, Program Latency=350 µs, Erase Latency=1.5 ms

Instead, FTL assigns a logical page to any free physical page of the entire SSD in a round-robin fashion, balancing the load and better exploiting the parallelism of the SSD architecture. A dynamic mapping strategy provides superior performance improvement in comparison with static mapping strategies [51]. Besides, FTL triggers the background GC process when the ratio of free pages within a plane drops below 40%. For the flash chips, we use the timing and organization characteristics of a real 256 Gbit (32 GB) SLC NAND product [37], summarized in Table 1.

Shared Channel Model. To exactly model the timing and strength of the NAND drivers located at FCCs of the shared channels, we use the nominal resistance and capacitance options of the considered flash product (given in [37]).

Workloads. Table 2 shows the selected real workloads from standard benchmark suits, reflecting high-end enterprise environments. TPCC [52] and TPCE [53] workloads are two groups of benchmark programs reflecting On-Line Transaction Processing (OLTP) applications, collected by Microsoft [1]. Two other OLTP traces, namely Financial1 and Financial2, were collected at two large financial institutions [3]. Exchange workload was collected on a Microsoft Exchange 2007 SP1 mail server for 5000 corporate users [1]. MSN traces were collected on an enterprise file server [2]. MSRC workloads were collected at Microsoft Research Cambridge data center servers previously characterized in [39]. WebSearch was collected from a popular internet search engine [3]. We also use a set of synthetic traces created by DiskSim's trace generator [9], to provide better understanding of the proposed communication infrastructure, .

Metrics. Our main performance metrics for the enterprise SSD evaluations are *average response time* and *maximum IOPS*. The response time is defined as the time elapsed from the arrival of a host I/O request until the response is sent back via SSD's host interface. For maximum IOPS calculation, we count the number of accomplished I/O operations in the unit of time when SSD is under full stress condition. To reach this condition, we ignore the inter-arrival time of I/O events and successively service them when there is at least one free slot in the host interface queue.

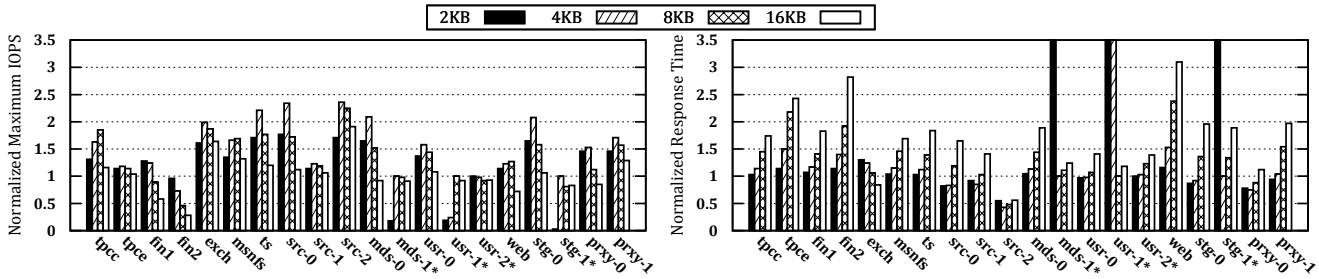


Figure 3: SSD performance vs. page size for different workloads. The results are normalized to those of 1 kB page size, except for workloads tagged with star (*) that saturate SSD in small page sizes because of their high rate of I/O requests (cf. Table 2). The figure shows how the employment of page size larger than 4 kB degrades the overall performance of SSD.

Table 2: Characteristics of the evaluated I/O workloads.

Trace	Read Ratio	Req. Size[a] Mean/Mode	Inter Arrv.[b]	Workload Description
tpcc	0.67	8.2 / 8	0.29	Microsoft TPC-C (OLTP)
tpce	0.94	8.1 / 8	0.45	Microsoft TPC-E (OLTP)
fin1	0.23	3.4 / 3	8.19	Financial1 (OLTP)
fin2	0.82	2.4 / 0.5	11.08	Financial2 (OLTP)
exch	0.28	15.9 / 8	1.05	Microsoft Exchange server
msnfs	0.67	10.1 / 8	0.51	MSN storage file sever
ts	0.18	9 / 4	387.95	Terminal server at MSRC
src-0	0.11	7.2 / 4	448.63	Source control 2-0 at MSRC
src-1	0.98	59.3 / 64	1044.68	Source control 2-1 at MSRC
src-2	0.3	56.3 / 64	547.66	Source control 2-2 at MSRC
mds-0	0.12	9.2 / 4	499.41	Media server 0 at MSRC
mds-1	0.93	56.8 / 64	366.46	Media server 1 at MSRC
usr-0	0.4	22.7 / 4	270.25	User home dirs 0 at MSRC
usr-1	0.91	49.5 / 64	13.36	User home dirs 1 at MSRC
usr-2	0.81	43.8 / 64	57.22	User home dirs 2 at MSRC
web	0.99	15.2 / 8	2.99	WebSearch
stg-0	0.64	40.8 / 64	297.79	Web staging 0 at MSRC
stg-1	0.15	11.6 / 4	275.27	Web staging 1 at MSRC
prxy-0	0.03	4.8 / 1	48.16	Web proxy 0 at MSRC
prxy-1	0.65	12.6 / 8	3.58	Web proxy 1 at MSRC,
synth	0,0.5,1.0	0.5 to 1024	1.00	Synthetic, from DiskSim

[a] Kilobytes [b] Milliseconds

4. MULTI-CHANNEL SSD SCALABILITY

During last two decades, the improvements in flash memory technology have led to different effects on storage devices: bit density increases, but other important features including performance, lifetime, and energy are drastically reduced. On the other hand, the SSD vendors strive to provide higher capacity products while keeping their performance at a reasonable level [20, 49]. Accordingly, scaling the SSD capacity necessitates proper scaling of I/O performance to keep flash memory utilization at the highest level. SSD scalability is always a matter of debate and recently attracted interests in academic and industry [14, 20, 21, 49].

The SSD storage hierarchy consists of four levels: 1) plane, 2) die, 3) chip, and 4) channel, and any attempt to scale SSD should target increasing capacity at one or more of them. In this section, we discuss SSD scalability behavior with respect to increasing storage capacity at each level. To elucidate how much performance improvement is due to the higher capacity or parallelism and how much is the contribution of process technology, we assume a fixed feature size (20 nm) and focus on SSD scalability.

4.1 Increasing Plane Capacity

The finest way to increase plane capacity is page size expansion by embedding more memory cells or increasing bit storage density of each cell. NAND flash is subject to wear-out failure and requires some types of ECCs to achieve a certain level of reliability. Thus, as the page size increases, a strong ECC is required that imposes considerable latency overheads to read and program operations. On the other side, increasing bit density to two or more bits per cell significantly reduces read/program performance because of slow circuit-level mechanisms required for correct detection/adjustment of memory cell's voltage level. For instance, Grupp *et al.* showed that density increase from 1 to 3 bits/cell will result in a performance degradation factor of 3–10 for I/O requests [21].

At architecture level, page size expansion has advantages and disadvantages. Larger page size results in lower number of executed operations, as on each access a large amount of data chunks are brought in or out. However, this performance enhancement is saturating and will be increased up to a point that the unit of host I/O requests equals the NAND flash memory page size. Increasing page size beyond this threshold is not beneficial and even can result in declined performance due to the partially-updated pages. Partial updates require read-modify-write operations (reading the stored page content, transferring to the controller, and merging with the updated data) and increase the chip-waiting time as the controller should wait for a busy chip to become idle in order to read a page. In addition to the above disadvantage, the transfer time of page data increases proportional to the page size. For read operations, this is extremely challenging since transfer time is close to the command execution time and this overhead may diminish the savings in number of executed commands [49].

Figure 3 illustrates the effects of page size on the performance of a SSD with configuration mentioned in Section 3. The results are normalized to those of 1 kB page size, except for the workloads where internal SSD traffic load saturates for small page sizes; in these cases (marked with *), the smallest page size (>1 kB) with unsaturated results is used for normalization. In general, the maximum SSD IOPS stops increasing at 4 kB page size and then starts decreasing owing to higher transfer time of a huge page and frequent partial updates. To have a better understanding, Table 3 reports the *ratio of partial updates*, *average read/program operation count per I/O request* and *average internal transfer size* for different pages sizes. The results confirm the increasing/decreasing trend of IOPS in Figure 3. We see that ratio of partial updates and resultant average transfer size sharply increase as page size exceeds 4 kB. So, the per-

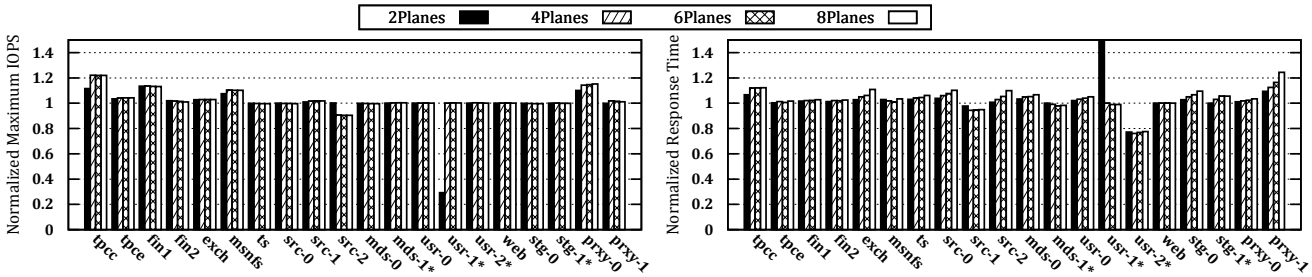


Figure 4: SSD performance vs. number of planes per die for different workloads. While the IOPS of target SSD remains roughly constant, response time increases in 6 or 8 planes per die.

formance improvement due to reduction in the number of I/O operations is not large enough to mitigate the performance loss of the increased partial updates and transfer size. We observed a similar behavior for SSD's response time, and depending on the application, the response time steadily increases from 2 kB page size to 4 kB, 8 kB or 16 kB. In sum, scaling SSD's capacity at finest granularity (page size) seems to result in lower I/O traffic, but significantly suffers from partial update and transfer size overheads.

We can also enlarge plane size by embedding more data pages. As a plane is the smallest unit to process a page request and has no means for internal parallelism, adding pages to a plane gives no performance gain if it does not increase bit-line latency. Therefore, this approach does not brings better capacity and performance for scalable design.

4.2 Increasing Die Capacity

We can increase SSD capacity by adding more planes to a die and expect that plane-level parallelism helps us to stripe requests for better performance. In theory this is correct, but extreme use of plane-level parallelism confronts serious limitations in practice: 1) As described in Section 2, plane-level commands have to be applied at the same page address of planes within a die. Then, finding requests in the controller queues that can be handled at same addresses of planes in a die is not always successful; 2) The allocation scheme has significant impact on the chance of finding multi-plane commands. Hu *et al.* [25] showed that the static allocation with channels/dies striping in priority and the dynamic allocation give best performance results for current SSDs. However, both schemes reduce the chance of executing multi-plane commands. Indeed, they empirically showed that when dynamic allocation scheme is employed in a multi-channel SSD, multi-plane commands are not required since the exploration of channel-level and chip-level parallelism alone can deal with majority of the requests. Then, reordering, rescheduling and classification algorithms, like PAQ [27] or the proposed schemes in [10, 40], have a very low chance to find multi-plane commands.

To explore the effect of plane-level parallelism on SSD performance, we assumed a maximum of 8 planes per die and conducted simulations for a SSD with configuration in Section 3. The experiments are for dynamic allocation scheme [25] and PAQ scheduling [27]. Figure 4 demonstrates the maximum IOPS and average response time for evaluated workloads normalized to the configuration with 1 plane per die. The results confirm that adding planes to a die may improve IOPS since the average number of executed operations per die can be increased through multi-plane commands.

Table 3: The effect of pages size increase on the executed commands in SSD evaluated in Figure 3.

Page Size (kB)	**1**	**2**	**4**	**8**	**16**
Partial Update Ratio	0.02	0.06	0.09	0.21	0.30
ReadOp/Req.[a]	16.1	8.4	4.4	2.7	2.1
ProgramOp/Req.[a]	21.6	11.1	5.8	3.3	2.4
Xfer/Req. (kB)[b]	37.7	39.0	40.8	48.0	72.0

[a] Read/program operation count per I/O request
[b] Internal transfer size per I/O request (kB)

Nevertheless, the improvement rate is almost negligible for half of the workloads and it usually remains unchanged for the die size of $\geq$ 4 planes. Figure 4 also shows that improving plane-level parallelism generally increases the average response time. Even though there are exceptions, this trend can be justified by the interactions between dynamic page allocation scheme and multi-plane command execution [25]. We conclude that having 2 or 4 planes per die is the most proper design choice for majority of workloads, beyond which performance can be degraded substantially.

Based on the above discussions on capacity increment at plane-level and die-level, we should say that intra-die scaling approaches are all limited by performance requirements of enterprise SSDs. Accordingly, the inevitable solution is to employ more dies or chips to increase capacity while fulfilling performance demands through die interleaving, flash chip pipelining, and channel striping by proper page allocation strategies [28, 49]. State-of-the-art enterprise SSD products follow the same approach to achieve true scalability [18, 26].

4.3 Increasing Flash Chip Capacity

One may consider increasing chip capacity by adding more dies and hence improving the chance for interleaving command execution to concurrently read, program, and erase pages and blocks of different dies. Moreover, interleaving provides an opportunity to overlap data transfer to a die with command execution of other dies within the same chip. Figure 5 shows this fact by giving the percentage of time that a flash chip spends in either of the following four states:

1. Exclusive transfer: one die is transferring data while others are waiting for data transfer or are idle.
2. Overlapped transfer and command execution: some dies are executing command and some are transferring data. The remaining dies are idle.
3. Exclusive command execution: one or multiple dies are executing commands while others (if any) are idle.
4. Idle: all dies are idle.

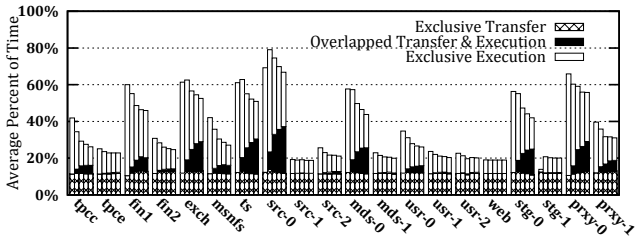


Figure 5: Average percent of time that each flash chip spends in data transfer or execution states. For each workload, number of dies per chip is set to 1, 2, 4, 6, and 8 from left to right.

In this experiment, we use the SSD configuration in Table 3 and varying number of dies per flash chip from 1 to 8. The figure shows that the time percentage that a flash chip exclusively transfers data to/from a die remains nearly constant regardless of the number of dies and workload. This is because of the technique used to conduct die interleaved commands which issues a command when all dies within the flash chip are idle [25]. In addition, data or command information transfer is triggered at the same time but in a sequential order. Data transfer and command execution of multiple dies within a flash chip can be more overlapped as die-level parallelism increases. Nevertheless, it stops after some level of parallelism above which it is not practically possible to improve chance of overlapping. The reason is that dies within a chip maximally utilize the available bandwidth for the chip and percentage of time, in which at least one die is transferring data, becomes nearly fixed (exclusive transfer + overlapped transfer and command execution in Figure 5).

Therefore the overall disk throughput is expected to improve as the die-level parallelism increases, but it stops improving after a certain level of parallelism. Figure 6 reports the maximum IOPS and average response time for different number of dies per flash chip. The results are given for 2 dies to 8 dies per chip configurations, normalized to the results for the configuration with 1 die. We remark that dynamic allocation strategy prefers to execute interleaved commands for higher die utilization, leading to higher throughput but in cost of worse response time [25]. Then, increasing flash chip capacity by means of excessive die interleaving is not a proper solution and is inherently limited by the channel bandwidth.

4.4 Increasing Number of Flash Chips

The primary option to increase SSD's capacity is to employ more flash memory chips. Today's state-of-the-art enterprise SSDs consist of more than 30 flash memory chips arranged on single-layer or multi-layer boards [18, 23, 26]. In a multi-channel configuration, more flash memory chips can be added either by enhancing channel-level parallelism (i.e., more shared channels) or by increasing chip-level parallelism (more flash chips sharing a channel). Increasing channel-level parallelism can be used to maximize independent connections to the controller, thereby minimizing the total cost of data and command transfer, performancewise. As stated in Section 2.2, FCCs of enterprise SSDs are usually dedicated to channels and are all embedded into the same FPGA/ASIC controller chip. Therefore, adding more channels necessitates rethinking of the controller logic and pin assignment that is costly. This is why 8-channel controller is a common configuration in many current products [35, 45] and adding more channels requires some hardware accelerators or special design techniques to drive huge number of resources [42, 43].

Added flash chips can be assigned to channels and expect that chip-level parallelism helps us to freely distribute read/program operations. The greater the number of connected flash chips to a shared channel, the higher capacitive load on the bus and lower performance. So, SSD standards like ONFI are always putting strict constraints on maximum load of a channel in order to deliver highest bandwidth. For example, ONFI 3.2 [7] allows maximum capacitance of 32.7 pF for I/O pins to guarantee 533 MB/s transfer rate when 8 flash chips are connected to a channel.

4.5 Summary

Despite the fact that increasing the number of functional units (chips or dies) is an inevitable approach to achieve capacity scaling, our analysis in last two subsections indicate that the limited bandwidth of channels is a potential showstopper to attain scalability goals. To this end, researchers and manufacturers look for flexible and high performance ways to cope with this problem. Some vendors such as Intel [26] and Fusion-io [18] use a high-level solution in which a set of small-sized SSD modules in conjunction with a front-end request distributor are used to produce a high-capacity product. The front-end chip may be a RAID controller or just a simple bridge. The 800 GB version of Intel 910 Series is a well-known example in this line which constitutes an on-chip PCIe-to-SAS bridge together with four 200 GB SSD modules each containing its own ASIC controller [26]. Even though such solution keeps the flash channel interface unchanged, it dramatically increases the cost of SSD controller due to the additional front-end logic and per module controller. Moreover, the number of failure points are increased in SSD and a single failure of a small-sized module brings the whole array down. Some vendors pay the extra cost and capacity penalties of RAID1 or RAID5 to alleviate this problem while others provide each module as an independent disk, i.e., there is no a single disk and customer must relay on software RAID techniques to achieve a single volume.

On the contrary, changing the flash communication interface may be a reasonable solution to alleviate bandwidth constraints of multi-channel bus structure. Using a shared and segmented network communication structure is a straightforward and promising alternative. For instance, HLNAND introduced a fully-packetized command and address format to enable connecting flash chips to the controller through a daisy-ring topology [46] or hierarchical ring topology [20]. Recently, Tavakkol *et al.* proposed a more generic and regular network-based design paradigm, namely Network-on-SSD (NoSSD) [50], which showed significant performance over conventional designs. In a similar manner, we suggest to use such a network-based solution and extend the preliminary NoSSD model for building large-scale SSDs. In the next section, we go over the details of NoSSD architecture and support it with messaging and communication protocols for inter-flash SSD communications.

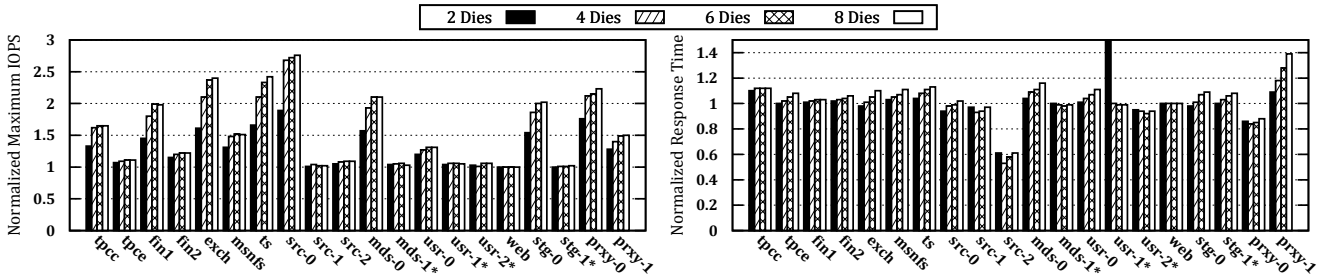


Figure 6: SSD performance vs. number of dies per flash chip for different workloads. While the IOPS remains a constant at highly-loaded channels, response time drastically increases (between 10% to 40% for various configurations).

5. INTER-FLASH NETWORKING

Network-on-SSD consists of routing nodes and communication links to connect adjacent flash chips following some specific topology. NoSSD is inspired by the proven concepts and techniques from on-chip and off-chip interconnection networks in multiprocessors [15]. NoSSD is different from layered communication abstraction models and is orthogonal to flash-related command execution and communication decoupling techniques. In this section, we first present a component-based view of NoSSD and introduce the basic building blocks of a typical NoSSD. Then, we look at system-level architectural issues relevant to NoSSD design and enterprise storage.

5.1 A Simple NoSSD Example

Figure 7 shows a sample NoSSD structured as a grid supporting controller-to-flash-chips communications. Instead of the shared and dedicated channels, a more general scheme is adopted, employing a grid of routing elements spread out among flash memory chips, interconnected by communication links. At the first glance, a tree-like network seems great due to higher traffic rate at the controller side. However, this topology provides less inter-node route parallelism compared to grid topology in addition to other design constraints that lead us to use grid topology: 1) in dynamic page mapping, where a logical page address has the highest freedom to be mapped at any flash chip/die/plane, garbage collection needs page movement between flash chips. The grid topology greatly helps in this issue by providing some direct paths between chips that does not necessarily pass through FCC; 2) special FCC to network connection methods (see Section 5.4), that decrease the average FCC to flash chips distance, best suite the grid topology; 3) FCC can use higher number of regularly-patterned connections to the network since the pin-out/logic complexity of NoSSD is greatly simplified in grid network compared to bus and tree.

We adopt a simplified perspective where NoSSD contains the following fundamental components.

Flash memory network adapters. They implement the interface by which a flash memory chip connects to NoSSD. Due to structural differences between network transmission unit (i.e., packet) and NAND flash memory command execution data (i.e., page or command), the network adapter should manipulate and arrange data for both sides. A good network adapter must provide a low latency path to the network while incurring no cost to commodity NAND flash.

Routers. They route data based on a chosen protocol and flow control scheme. Details of the input-buffered router element alongside flash memory logic are shown in Figure 7. We suggest to use 4 injection/ejection channels per router to reduce waiting time of the sent/arrived packets.

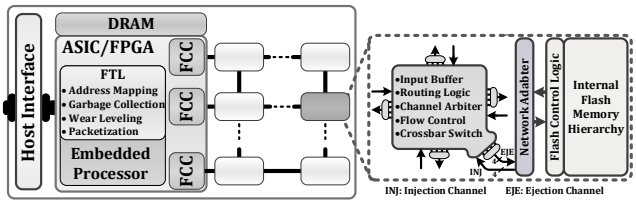
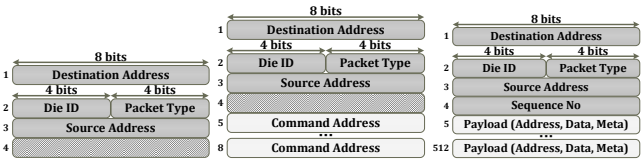


Figure 7: Topological illustration of NoSSD.



(a) 4 flits (b) 8 flits (c) 512 flits

Figure 8: Packet format for different packet types in Table 4.

Links. They connect the routers and flash controller and provide the raw bandwidth. To maintain the wiring costs low and alleviate pin-out constraint, links are bidirectional.

FCC (redesigned). Shared-buffer interfaces are used to connect SSD controller to NoSSD. These interfaces are tuned to the messaging protocol and flow control scheme. In fact, FCC simply emulates network adapter functionalities.

Figure 7 shows a high-level view of the NoSSD. To realize NoSSD, detailed *messaging protocol*, *flow control and routing mechanism* and *controller placement* must be considered.

5.2 Packet Format and Messaging Protocol

As described above, the network adapter decouples the flash chip and the controller from NoSSD. In other words, network adapter handles encapsulation of NAND flash messages, generated by either the controller or flash chips, for the routing strategy. A NAND flash message consists of command, data, and metadata that is broken into packets for transmission over NoSSD. At the lowest transmission level (i.e., link), the basic datagram are flits (atomic flow control units forming a packet). In NoSSD, we assume that flit is the minimum size of datagram that can be transmitted in one link transaction; it is set to 8 bits for our NoSSD model in Section 6.

Figure 8 shows the packet formats of NoSSD consisting of

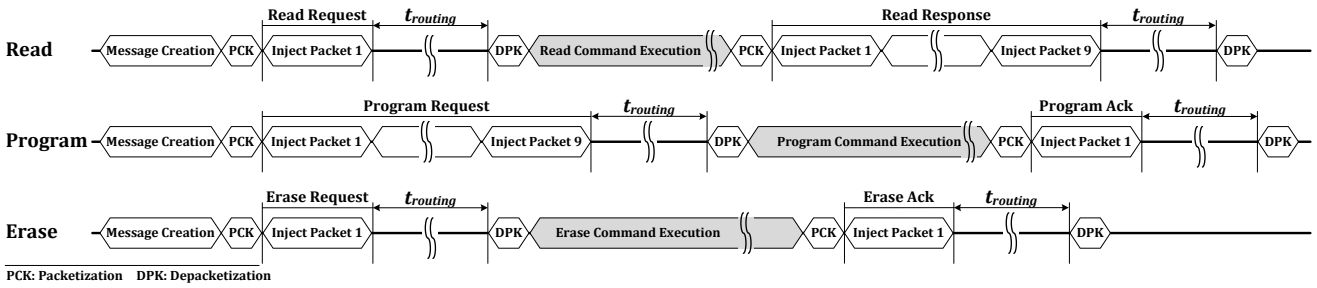


Figure 9: NoSSD messaging protocol for basic NAND flash operations.

Table 4: Binary encoding and length of packet types. The reserved binary codes are for advanced commands.

Type	Code	Len.	Type	Code	Len.
Read Req.	0000	8	Erase Req.	0100	8
Read Resp.	0001	512	Erase Ack.	0101	4
Program Req.	0010	512	Reserved	1xxx	x
Program Ack.	0011	4			

header flits followed by payloads. The header part includes source and destination addresses and packet sequence number. Two additional 4-bit units are used to identify the type of a packet and ID number of the target die. The binary encoding of the packet types is presented in Table 4 based on which packet length is also determined. Each packet belonging to the same message has the same ID number to differentiate it from other packets, when it reaches the controller or flash chips. Upon receiving packets at the destination, the network adapter uses packet sequence number to depacketize the received command or data.

Figure 9 illustrates the messaging protocol to conduct basic command execution of the flash chip. To initiate a read operation, the controller encapsulates the command and target page address into a message which is then packetized to a single 8-flit *Read Request* packet (Figure 8b). Having the read command executed, the flash chip produces a message containing the data and metadata of the read page. Assuming 4 kB page and 224 B metadata, as considered in Section 3, this message is sent to the controller through nine 512-flit *Read Response* packets (Figure 8c). To issue a program operation, on the other hand, the controller generates a message, containing the page address, data, and metadata, that is broken into nine 512-flit *Program Request* packets. Upon finishing program command execution, the target flash chip generates a short 4-flit *Program Ack* acknowledge packet (Figure 8a). A similar mechanism is used during erase operation by a pair of *Erase Request* and *Erase Ack* packets.

5.3 Router Organization and Functionalities

In this section, we describe the router organization of NoSSD and its corresponding functionalities.

NoSSD Router. Figure 7 clearly specifies the major components of a router in NoSSD including buffers, crossbar switch, routing and arbitration unit, and link controller. The switch connects the input buffers to output ports, while routing and arbitration unit implements the algorithm dictating these connections. Moreover, the routing logic is separately implemented for each input port; so, it can simultaneously forward maximum number of flits per input/output switching pairs to reduce routing delay and enhance communication bandwidth. Based on this model, NoSSD routers use a pipelined synchronous architecture while intra-router data transfer is handled asynchronously using the link controller.

Wormhole NoSSD. In this work, NoSSD utilizes wormhole switching mechanism merely for transport of data, and the routing mechanism for determining the path of data transport. NoSSD guarantees lossless packet delivery using wormhole switching which prevents buffer overflow at input/output ports [15]. Wormhole combines packet switching with the data streaming quality of channel pipeline communication to attain a minimal packet latency. At each hop, router looks at the header to determine the proper outpot port and immediately forwards it; subsequent flits are forwarded as they arrive. This causes the packet to experience an extremely reduced amount of network transfer latency and, as a consequence, the overall NoSSD performance is improved. When the path is determined, ahead of crossbar switch, each flit has to wait for availability of free slots in the next hop buffer and then preempts output channel via arbitration. At last, flits are transmitted over physical links. When header reaches the destination, the local ejection port is allocated for flit-by-flit delivery of the packet to network adapter.

ON/OFF Flow Control in NoSSD Routers. To ensure a correct and lossless operation, NoSSD utilizes ON/OFF wormhole flow control mechanism where each downstream router uses a status signal to inform the upstream side about the availability of free buffer slots. While status signal is ON and there are flits waiting for dispatch, the upstream router keeps on flit transmission. However, it stops sending flits upon status turn off since buffer slot shortage will cause flit loss on the downstream side. In practice, this signal is turned off when the number of free slots falls below a predetermined threshold value. This value should be chosen to guarantee storage of on-the-fly flits while signal is propagated back to the upstream side. Consequently, there is a lower limit for the minimum buffer size of each input port that must be considered during NoSSD design. Regarding the router model in Section 6, the minimum buffer size is 10 flits based on the rules given in [15].

NoSSD uses bidirectional channels to alleviate pin-out constraints on both sides of FCC and flash chips. A channel demands two ON/OFF signals for each direction and a token signal, namely Transfer Token, is shared between adjacent routers to manage channel accesses and avoid contentions. One side is allowed to transfer when the token signal is low

while other side sends flits when Transfer Token is high. Upon finishing flit transmission, the sender inverts the token value to grant access permission to the opposite side.

The ON/OFF flow control and wormhole switching guarantee in-order delivery of flits within a packet. However, multi-path and prioritized routing, described in section 5.4, may cause out-of-order delivery of the packets within a message. Nonetheless, the packet sequence number can be simply used for accurate recovery of the original message.

5.4 NoSSD Design Rules for High Performance SSDs

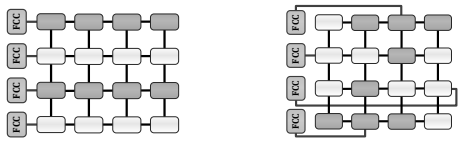
Priority-based Transmission. Performance of NoSSD may suffer from long transmission latency of *Read Response* and *Write Request* packets. In fact, in wormhole switching, a stalled long packet may cause serious side effects on the transmission latency of other packets due to blocking propagation.

To mitigate this shortcoming, NoSSD routers exploit a packet prioritization scheme for arbitration on the accesses to output channels. To this end, router micro-architecture is slightly modified in order to alter the normal FIFO arbitration policy and give higher priority to some packets (based on some rules). To explain these rules, we classify packets into two groups: small packets including *Read Request*, *Program Ack*, *Erase Request* and *Erase Ack*, and large packets including *Read Response* and *Program Request* packets.

Rule 1. ***Small packets are always handled first.*** *Following this policy, transfer delay of small packets is reduced but waiting time of large packets is increased. We expect that the performance improvement achieved by accelerating the transmission of small packets is large enough to compensate for the latency loss of large packets. This rule does not result in starvation for large packets since generation of small packets always depends on NAND flash operation execution; hence, generation rate of a burst of small packets is predictably low.*

Rule 2. ***Transmitting a sequence of large packets are prioritized.*** *When NoSSD uses XY routing, Read Response or Program Request packets of the same message are always routed on the same path. Therefore, if a large packet is granted to use a channel, we can keep it for another upcoming packet (that will be received at the same port and with same die ID) to be transmitted immediately. Therefore, read/program data will be always transmitted in chunks that improve response time. This approach generally differs from circuit switching since large packets may stall at intermediate routers when there are many requests in the network.*

Traffic-Aware Packet Injection. On write transactions, FTL triggers an allocation process to choose free pages for data storage. Static allocation tries to uniformly distribute page writes among flash chips following a predetermined mapping rule; that is target flash chip, die, and plane address is determined based on logical page address. Dynamic allocation, on the other hand, assigns pages dynamically by considering different factors such as idle/busy state of the channels and flash chips and priority order of parallelism. At the first step, FTL searches for a free physical channel based on round-robin approach. Next, within the allocated channel, it assigns a free flash chip and then allocates die and plane using the same round-robin method. Although dynamic allocation provides better performance, it suffers from degraded random read latency [24]. Actually, allocations considering write performance and cell wearout result in a non-uniform distribution of read operations that may not be tolerable. To relieve read latency degradation, the controller in the ultimate NoSSD design, is augmented with a simple traffic-aware injection mechanism. Each FCC has a counter referring to the number of packets waiting to be sent. Depending on the counter values, controller chooses the less-stressed FCC for packet injection.



(a) One Sided (b) All Sided

Figure 10: One- vs. all-sided FCC-to-NoSSD connections.

Controller Placement. The NoSSD design in Figure 7 uses a grid topology and has FCCs positioned near to the left side of network. However, keeping NoSSD regularity in mind, there are multiple other ways to connect FCCs to NoSSD routers. These different configurations of the FCC-to-NoSSD connection can have a dramatic impact on the latency and bandwidth characteristics of the network, especially for grid topology which is not edge symmetric. Furthermore, by reducing the variance in the number of hops per request as well as packet latency and channel load, NoSSD's performance seems to be less sensitive to the flash chip on which a page is mapped. In the ultimate NoSSD design that is evaluated in Section 6, we examine one-sided and all-sided FCC placement policies, as shown in Figure 10. In all-sided configuration, we try to provide better and predictable latency and bandwidth characteristics by evenly distributing FCCs in all sides of NoSSD (middle of each side). Please note that the longer wires of all-sided placement still have lower latency with respect to the multi-channel bus wires since each output driver, either at FCC or NoSSD side, sees much lower capacitive load.

5.5 Implementation Issues

In a conventional multi-channel architecture, a flash controller consists of two parts: 1) a unit responsible for controlling the sequence of micro-operations required for NAND flash functionalities and error correction schemes, and 2) a signaling mechanism for transmitting controller commands to flash chips. In NoSSD controller, on the other hand, managing read/write requests are accomplished by the flash chips themselves. Thus, controller's functionality is limited to be the interface logic between the front-end processor and network substrate. This simplifies controller logic which is highly desirable in modern SSDs[2].

In a NoSSD flash chip, the network adapter, buffer storage, crossbar switch, channel arbiters, and router logic are the newly added parts. Among the mentioned elements, buffers occupy most of the router's area. To mitigate signal propagation delay and fully utilize channel bandwidth, we assume buffer depth of 512 flits per input channel which means a 4 kB buffer space per router. To estimate the area

[2]Many companies try to embed some memory-related functions (like error correction) into the NAND flashes so that the controller cost is reduced; an example is Managed NAND by Micron [36].

Table 5: Characteristics of the simulated NoSSD model.

NoSSD
Channel Width = 8 bit, Inter-router Propagation Delay = 5 ns, Pipeline Stages = 3, Pipeline Stage Delay = 5 ns, Input Buffer Size = 512 flit, Routing Algorithm = XY

overhead of the router, we used Orion 2 [29]. We set process technology to 32 nm, as it is the smallest available choice in Orion 2 that matches ever shrinking feature size of NAND flash memory. The results show that the buffer and crossbar area is about 0.17 mm^2 and 0.03 mm^2, respectively, while the arbitration and routing logics just occupy 0.002 mm^2. Next, we use NVSim [16] to achieve an estimation for the area of a typical 4 GB NAND flash die with the properties mentioned in Table 1. The results show that the die area is about 72 mm^2 and thus a set of 4/8 dies within a 16/32 GB flash chip occupy 288/576 mm^2. In other words, the area overhead of a NoSSD router is less than 0.1% of the flash storage area. Please note that due to model limitations, we ignored some components in each part, i.e., the NAND peripheral and command control circuits for memory and the network adapter for NoSSD.

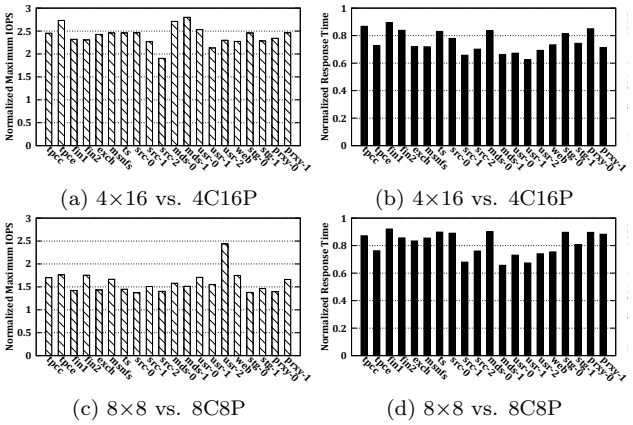
Regarding signaling requirements, NoSSD also simplifies FCC implementation logic since each communication channel is managed via 3 control signals regardless of the number of flash chips. Indeed, FCC in a multi-channel architecture has to drive tens of control signals that is proportional to the number of dies sharing a channel [25]. In addition, as mentioned in the NAND flash chip standards [5, 6], SSD designers should care about capacitive and resistive load of a bus channel for reliable and high-performance transfers [6]. NoSSD, nevertheless, can remove these electrical constraints since each channel is just shared between two adjacent routers and the global wiring is provided by (semi-)regular topologies, e.g., mesh (grid) and torus.

6. EVALUATION

To compare NoSSD's performance against a baseline multi-channel SSD design, we use the simulation methodology and flash chip model described in Section 3. For NoSSD, the flash chips are organized as a grid topology with XY routing [15]. As an abbreviation, we simply use $m \times n$ notation for a grid configuration of m rows and n columns. The multi-channel SSD structure with same size, m channels and n flash chips per channel, is denoted with $mCnP$. The NoSSD router is the same as model introduced by Dally and Towles [15] without virtual channel multiplexing. The router pipeline consists of three stages each with 5 ns latency: 1) buffer write and route computation for header flits, 2) switch allocation, and 3) switch traversal, followed by link traversal. Table 5 summarizes the evaluated router configuration. Please note that, the default packet injection policy in the FCC is round-robin.

6.1 NoSSD vs Multi-Channel SSD Performance

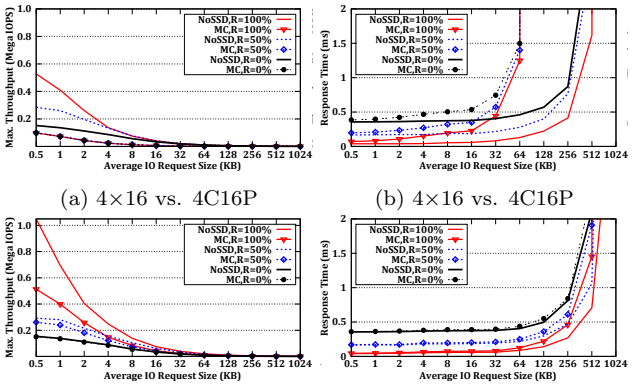
Figure 11 presents the normalized maximum IOPS and average response time of real workload traces in 4×16 and 8×8 NoSSD configurations with respect to the same-sized multi-channel SSD. We see that NoSSD enhances the maximum IOPS by 37%–180% (100%, on average) whereas it reduces the average and worst-case response time by 22% and 8%, respectively. In addition, Figure 12 illustrates the



(a) 4×16 vs. 4C16P (b) 4×16 vs. 4C16P

(c) 8×8 vs. 8C8P (d) 8×8 vs. 8C8P

Figure 11: Normalized average response time and maximum IOPS of NoSSD to the multi-channel baseline. MC stands for multi-channel SSD.



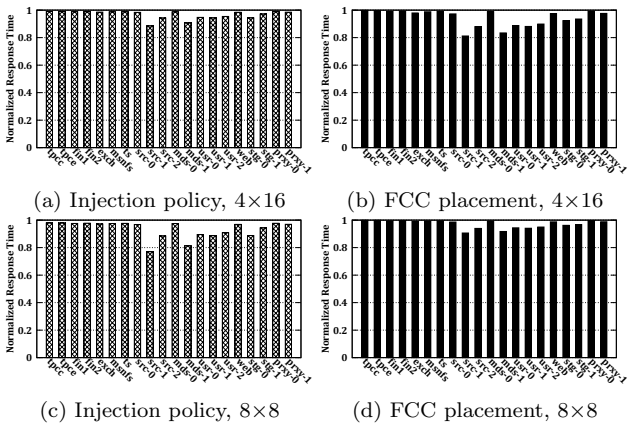
(a) 4×16 vs. 4C16P (b) 4×16 vs. 4C16P

(c) 8×8 vs. 8C8P (d) 8×8 vs. 8C8P

Figure 12: Response time and maximum IOPS of NoSSD and multi-channel baseline under synthetic workloads.

maximum achievable IOPS and average response time of the NoSSD and multi-channel SSD designs for different request sizes in synthetic workloads. Regarding the results of both real and synthetic workloads, we can make the following conclusions:

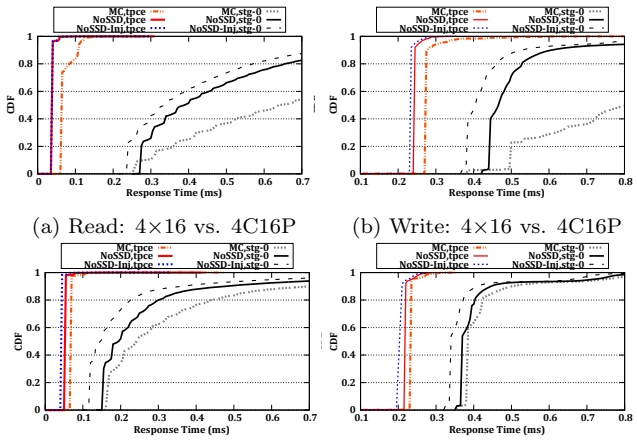
- NoSSD achieves much more performance improvement (compared to the conventional design) when channel-level parallelism is low. This is primarily due to better utilization of resources in NoSSD by pipelining/parallelizing requests among routers, while path diversity and round-robin packet injection collectively provide a substrate to balance load in the network.
- As the ratio of read requests increases in the workload, NoSSD achieves higher performance improvement. In fact, read-oriented workloads are more sensitive to the communication delays, since the execution time of a read operation is comparable with controller-to-flash communication delay.
- NoSSD provides great response time gains when workload request size is large. Contrarily, the greatest IOPS improvement is seen for smaller request sizes.
- By scaling SSD capacity via increasing channel-level par-



(a) Injection policy, 4×16 (b) FCC placement, 4×16

(c) Injection policy, 8×8 (d) FCC placement, 8×8

Figure 13: The effect of enhancing FCC to NoSSD communication (a),(c) via traffic-aware injection policy and (b),(d) via FCC placement.



(a) Read: 4×16 vs. 4C16P (b) Write: 4×16 vs. 4C16P

(c) Read: 8×8 vs. 8C8P (d) Write: 8×8 vs. 8C8P

Figure 14: Cumulative distribution function (CDF) of the read and write response time of the two representative workloads, i.e., `tpce` and `stg-0`, for NoSSD with normal injection (NoSSD), NoSSD with traffic-aware injection (NoSSD-Inj) and multi-channel SSD (MC).

allelism, the overall performance gain of using NoSSD decreases (higher response time and lower IOPS values). Comparing results of 4×16 and 8×8 in terms of response time shows that maximum increase in overall response time is about 10% for `prxy-1`. In all other applications, however, response time increase of high channel-level parallelism is very small (4%, on average). For maximum IOPS results, the reason for larger throughput improvement in 4×16 NoSSD compared to 8×8 NoSSD is two-sided. First, the dynamic allocation scheme well uses higher channel-level parallelism in $8C8P$ baseline and hence NoSSD has less chance to improve performance. Second, the 4×16 configuration roughly doubles the chip-level parallelism and NoSSD can better pipeline flash operation execution over adjacent flash chips.

6.2 Performance of Packet Injection Policy and FCC Placement

So far, we have shown performance improvement results for NoSSD, **without** considering design considerations of packet injection and FCC placement. For the sake of completeness, Figure 13a and Figure 13c illustrate the response time of NoSSD with the traffic-aware packet injection policy, normalized to that of the same size NoSSD with the normal (round-robin) injection policy. As can be seen, the maximum performance improvement using this policy is 23% (in case of `src-1` in 8×8 NoSSD) while it is negligible in most of the other workloads. On the average, injection policy gives an aggregate response time improvement of 7% and 4% for 8×8 and 4×16 NoSSDs, respectively. Nevertheless, we deduce that traffic-aware packet injection policy achieves better response time improvement for workloads with large average request sizes (`src-1`, `src-2`, `mds-1`, `usr-0`, `usr-1`, `usr-2`). The reason is that such workloads considerably benefit from the path diversity provided by the traffic-aware injection policy.

Figure 13b and Figure 13d show the response time of 8×8 and 4×16 NoSSDs with all-sided FCC placement, normalized to the same-sized NoSSD with one-sided placement. As the results show, all-sided FCC placement can reduce response time by up to 19% (and 5%, on average) for the evaluated configurations. Again, we can see that NoSSD performance is sensitive to the FCC placement when the average request size of the workload is large.

6.3 Response Time Variation

One of the major challenges with current multi-channel SSD architectures is their large variance of response time. This variability is specially harmful for streaming and time-critical applications where a deterministic response time must be guaranteed. To mitigate this variation, current enterprise SSDs rely on buffering methods and reconfigurable SSD controllers that result in more complex controller design. Figure 14 compares the response time variation of NoSSD and conventional SSDs by plotting the cumulative distribution function (CDF) of response time values. To increase readability, we just selected two representative applications with small request size (`tpce`) and large request size (`stg-0`). We see that the CDF curve of NoSSD sharply rises for `stg-0` (large request sizes better use resource sharing in NoSSD) and shows a considerable response-time predictability thanks to its path diversity and pipelined routing. For `tpce`, with small requests, the improvement is marginal.

7. RELATED WORK

Improving SSD performance and scalability for enterprise systems has been extensively explored in recent years. The main discussion is that since most of read requests are processed synchronously by OLTP and database system, even a single stall, caused by either slow programs, limited parallelism, or path latency of the controller, may considerably limit the overall SSD throughput [32, 33]. There are many studies trying to address this issue in high-throughput enterprise systems:

Enhancing channel-level parallelism. Enterprise SSD vendors improve capacity and I/O throughput by increasing the number of channels and employing suitable page allocation strategies [24, 34]. Although highly parallel channels relax resource contention, they substantially increase the cost of controller [31, 44, 48].

Utilizing intra-flash chip parallelism. Alongside channel level parallelism, FTL management policies can be designed to fully exploit parallelism provided at other levels of the SSD architecture [8, 10, 13, 22, 27, 40]. However, such solutions greatly suffer from additional latency and controller complexity required to attain their goals.

Design customization for enterprise workloads. An alternative strategy is to customize SSD architecture for efficient service of the requests in transactional workloads [26, 47, 48]. Similar to other solutions, architecture customization generally increases cost complexity of the controller. Recently, HLNAND proposed a scalable architecture which is built upon a unique daisy-ring architecture [46] or hierarchical ring topology [20]. In comparison with NoSSD, HLNAND products provide much less path diversity and their performance is still restricted by a single FCC at the controller side while NoSSD shares FCCs to get better performance/reliability. There are other examples of enterprise SSDs supporting a large number of flash chips at the backend. In contrast to NoSSD, these designs rely on hierarchal controllers (e.g. Intel 910 [26]) or hardware accelerators (e.g. PMC NVEMC [43]) at the cost of increased complexity and decreased reliability.

8. CONCLUSION

In this paper, we concentrated on the scalability challenges of the common enterprise SSD designs based on multi-channel communication architecture. Based on our evaluation, the intra-die scaling approaches are all prohibited by the performance requirements of enterprise SSDs. Therefore, the inevitable solution is to employ more dies in the design and exploit high-level (i.e., channel and flash chip) parallelism and request striping to enhance the overall SSD performance. However, simulation results show that higher level solutions are also limited by controller complexity and bus characteristics. Motivated by these observations, we proposed protocol and configurations for realizing a network-based communication structure (i.e., NoSSD), to achieve a high-performance SSD interconnect that is highly scalable and enjoys design regularity. The simulation results for NoSSD and its design considerations reveal higher performance, larger bandwidth, and more scalability when compared to the conventional design. We believe this paper is just an initial step in influencing SSD's scalability using architectural and interconnect enhancements. Considering the effect of different functionalities (e.g., garbage collection) and exploring different network topologies are of future research activities in this line.

9. REFERENCES

[1] Microsoft enterprise traces. `http://iotta.snia.org/traces/list/BlockIO`.

[2] Microsoft production server traces. `http://iotta.snia.org/traces/list/BlockIO`.

[3] UMass trace repository. `http://traces.cs.umass.edu`.

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

[5] JEDEC standard, NAND flash interface interoperability, Oct 2012.

[6] Open NAND flash interface specification 3.1, Sep 2012.

[7] Open NAND flash interface specification 3.2, Jun 2013.

[8] N. Agrawal et al. Design tradeoffs for SSD performance. In *USENIX ATC'08*, pages 57–70, Jun 2008.

[9] J. S. Bucy et al. The DiskSim Simulation Environment Version 4.0 Reference Manual. Technical Report CMU-PDL-08-101, Parallel Data Laboratory, Carnegie Mellon University, May 2008.

[10] A. M. Caulfield et al. Gordon: using flash memory to build fast, power-efficient clusters for data-intensive applications. In *ASPLOS XIV*, pages 217–228, Mar 2009.

[11] L.-P. Chang. On efficient wear leveling for large-scale flash-memory storage systems. In *SAC'07*, pages 1126–1130, Mar 2007.

[12] Y.-H. Chang et al. Endurance enhancement of flash-memory storage systems: an efficient static wear leveling design. In *DAC'07*, pages 212–217, Jun 2007.

[13] F. Chen et al. Essential roles of exploiting internal parallelism of flash memory based solid state drives in high-speed data processing. In *HPCA'11*, pages 266–277, Feb 2011.

[14] B. Collins. SAS SSDs - building blocks for high-performance enterprise storage. In *Flash Memory Summit*, Aug 2011.

[15] W. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2003.

[16] X. Dong et al. Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory. *IEEE TCAD*, 31(7):994–1007, Jul 2012.

[17] K. Eshghi and R. Micheloni. SSD architecture and PCI Express interface. In *Inside Solid State Drives (SSDs)*, volume 37 of *Springer in Advanced Microelectronics*, pages 19–45. 2013.

[18] Fusion-io, Inc. ioDriveIIDuo data sheet, 2013.

[19] E. Gal and S. Toledo. Algorithms and data structures for flash memories. *ACM Comput. Surv.*, 37(2):138–163, Jun 2005.

[20] P. Gillingham et al. 800 MB/s DDR NAND flash memory multi-chip package with source-synchronous interface for point-to-point ring topology. *IEEE Access*, 1:811–816, 2013.

[21] L. Grupp et al. The bleak future of NAND flash memory. In *FAST'12*, pages 17–24, Feb 2012.

[22] L. M. Grupp et al. The harey tortoise: Managing heterogeneous write performance in SSDs. In *USENIX ATC'13*, pages 79–90, Jun 2013.

[23] HGST, a Western Digital company. Ultrastar SSD800MM, enterprise solid state drives, 2013.

[24] Y. Hu et al. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In *ICS'11*, pages 96–107, May-Jun 2011.

[25] Y. Hu et al. Exploring and exploiting the multilevel parallelism inside SSDs for improved performance and endurance. *IEEE TC*, 62(6):1141–1155, Jun 2013.

[26] Intel Corporation. Intel solid-state drive 910 series, product specification, 2013.

[27] M. Jung et al. Physically addressed queueing (PAQ):

improving parallelism in solid state disks. In *ISCA'12*, pages 404–415, Jun 2012.

[28] M. Jung and M. Kandemir. An evaluation of different page allocation strategies on high-speed SSDs. In *HotStorage'12*, Jun 2012.

[29] A. Kahng et al. Orion 2.0: A fast and accurate noc power and area model for early-stage design space exploration. In *DATE '09*, pages 423–428, Feb 2009.

[30] J.-U. Kang et al. A multi-channel architecture for high-performance NAND flash-based storage system. *J. Systems Architecture*, 53(9):644–658, Sep 2007.

[31] D. Lavenier et al. Seed-based genomic sequence comparison using a FPGA/FLASH accelerator. In *FPGA'06*, pages 41–48, Dec 2006.

[32] S.-W. Lee et al. A case for flash memory SSD in enterprise database applications. In *SIGMOD'08*, pages 1075–1086, Jun 2008.

[33] S.-W. Lee et al. Advances in flash memory SSD technology for enterprise database applications. In *SIGMOD'09*, pages 863–870, Jun 2009.

[34] S. Liang. Algorithms designs and implementations for page allocation in SSD firmware and SSD caching in storage systems. Master's thesis, Computer Science and Engineering, The Ohio State University, 2010.

[35] Marvell. Marvell 88NV9145: Native PCIe gen 2.0 x 1 NAND flash controller, 2011.

[36] Micron Technology, Inc. Managed NAND.

[37] Micron Technology, Inc. MT29F256G08AECCB NAND flash memory, 2010.

[38] Micron Technology, Inc. Micron P420m PCIe SSD product brief, May 2013.

[39] D. Narayanan et al. Write off-loading: Practical power management for enterprise storage. *ACM Trans. Storage*, 4(3):10:1–10:23, Nov 2008.

[40] C. Park et al. Exploiting internal parallelism of flash-based SSDs. *IEEE CAL*, 9(1):9–12, Jan 2010.

[41] PMC-Sierra. 89HF08P08CG3 8-Channel x8 PCIe Gen3 enterprise NV-DRAM controller, May 2013.

[42] PMC-Sierra. 89HF16P04CG3 16-Channel x4 PCIe Gen3 enterprise flash controller, May 2013.

[43] PMC-Sierra. 89HF32P08CG3 32-Channel x8 PCIe Gen3 enterprise flash controller, May 2013.

[44] R. Rivera. Distributed data acquisition and storage architecture for the SuperNova acceleration probe. *IEEE Trans. Nuc. Sci.*, 55(1):246–250, Feb 2008.

[45] SandForce. SandForce SF-2000 flash storage processors, Apr 2013.

[46] R. Schuetz et al. Hyperlink nand flash architecture for mass storage applications. In *NVSMW'07*, Aug 2007.

[47] Y. J. Seong et al. Hydra: A block-mapped parallel flash memory solid-state disk architecture. *IEEE TC*, 59(7):905–921, Jul 2010.

[48] J.-Y. Shin et al. FTL design exploration in reconfigurable high-performance SSD for server applications. In *ICS'09*, pages 338–349, Jun 2009.

[49] A. Silvagni. NAND DDR interface. In *Inside NAND Flash Memories*, pages 161–196. Springer Netherlands, 2010.

[50] A. Tavakkol et al. Network-on-SSD: A scalable and high-performance communication design paradigm for SSDs. *IEEE CAL*, 12(1):5–8, Jan 2013.

[51] A. Tavakkol et al. Unleashing the potentials of dynamism for page allocation strategies in SSDs. In *SIGMETRICS '14*, pages 551–552, Jun 2014.

[52] Transaction Processing Performance Council (TPC). TPC benchmark^{TM} C, standard specification, Feb 2010.

[53] Transaction Processing Performance Council (TPC). TPC benchmark^{TM} E, standard specification, 2010.

[54] B. Van Houdt. A mean field model for a class of garbage collection algorithms in flash-based solid state drives. In *SIGMETRICS'13*, Jun 2013.