NANDFlashSim: High-Fidelity, Microarchitecture-Aware NAND Flash Memory Simulation

MYOUNGSOO JUNG, Yonsei University
WONIL CHOI, Pennsylvania State University
SHUWEN GAO, Intel
ELLIS HERBERT WILSON III, Panasas
DAVID DONOFRIO and JOHN SHALF, Lawrence Berkeley National Laboratory
MAHMUT TAYLAN KANDEMIR, Pennsylvania State University

As the popularity of NAND flash expands in arenas from embedded systems to high-performance computing, a high-fidelity understanding of its specific properties becomes increasingly important. Further, with the increasing trend toward multiple-die, multiple-plane architectures and high-speed interfaces, flash memory systems are expected to continue to scale and cheapen, resulting in their broader proliferation. However, when designing NAND-based devices, making decisions about the optimal system configuration is nontrivial, because flash is sensitive to a number of parameters and suffers from inherent latency variations, and no available tools suffice for studying these nuances. The parameters include the architectures, such as multidie and multiplane, diverse node technologies, bit densities, and cell reliabilities. Therefore, we introduce NANDFlashSim, a high-fidelity, latency-variation-aware, and highly configurable NAND-flash simulator, which implements a detailed timing model for 16 state-of-the-art NAND operations. Using NANDFlashSim, we notably discover the following. First, regardless of the operation, reads fail to leverage internal parallelism. Second, MLC provides lower I/O bus contention than SLC, but contention becomes a serious problem as the number of dies increases. Third, many-die architectures outperform many-plane architectures for disk-friendly workloads. Finally, employing a high-performance I/O bus or an increased page size does not enhance energy savings. Our simulator is available at http://nfs.camelab.org.

CCS Concepts: • **Computer systems organization → Architectures**

Additional Key Words and Phrases: Non-volatile memory, NAND flash memory, cycle-level simulation, solid state disk, performance evaluation

This paper is a full-length version of an earlier conference paper [Jung et al. 2012].
This research was supported in part by MSIP grants IITP-2015-R0346-15-1008 and NRF-2015M3C4A 7065645, DOE grant DE-AC02-05CH1123, and NSF grants 1213052, 1205618, 1302557, 1526750, 1409095, and 1439021. Myoungsoo Jung is the corresponding author.
Authors' addresses: M. Jung (corresponding author), School of Integrated Technology, Yonsei University, 85 Songdogwahak-ro, Yeonsu-gu, Incheon 21983, Korea; email: m.jung@yonsei.ac.kr; W. Choi and M. T. Kandemir, School of Electrical Engineering and Computer Science, Pennsylvania State University, 342 Information Sciences and Technology Building, University Park, PA 16802, USA; emails: {wuc138, kandemir}@cse.psu.edu; S. Gao, Visual and Parallel Computing Group, Intel, 1900 Prairie City Rd, Folsom, CA 956360, USA; email: shuwen.gao@intel.com; E. H. Wilson, Panasas, 1501 Reedsdale Street, Suite 400 North, Pittsburgh, PA 15233, USA; email: ellis@ellisv3.com; D. Donofrio and J. Shalf, Computational Research Division, Lawrence Berkeley National Laboratory, 1 Cyclotron Road, Berkeley, CA 94720, USA; emails: {ddonofrio, jshalf}@lbl.gov.

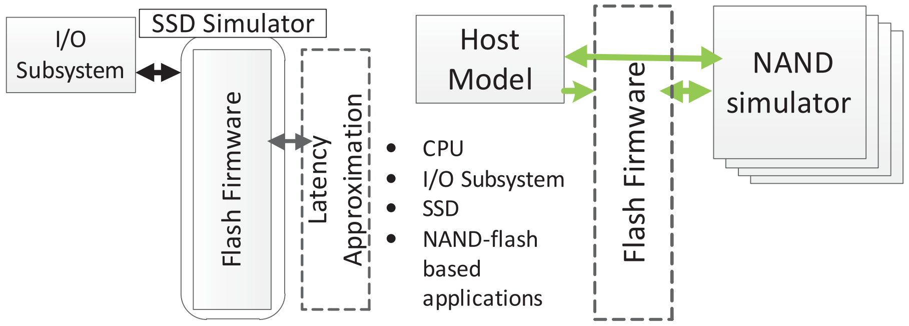
DOI: http://dx.doi.org/10.1145/2700310

ACM Reference Format:
Myoungsoo Jung, Wonil Choi, Shuwen Gao, Ellis Herbert Wilson III, David Donofrio, John Shalf, and Mahmut Taylan Kandemir. 2016. NANDFlashSim: High-fidelity, microarchitecture-aware NAND flash memory simulation. ACM Trans. Storage 12, 2, Article 6 (January 2016), 32 pages.
DOI: http://dx.doi.org/10.1145/2700310

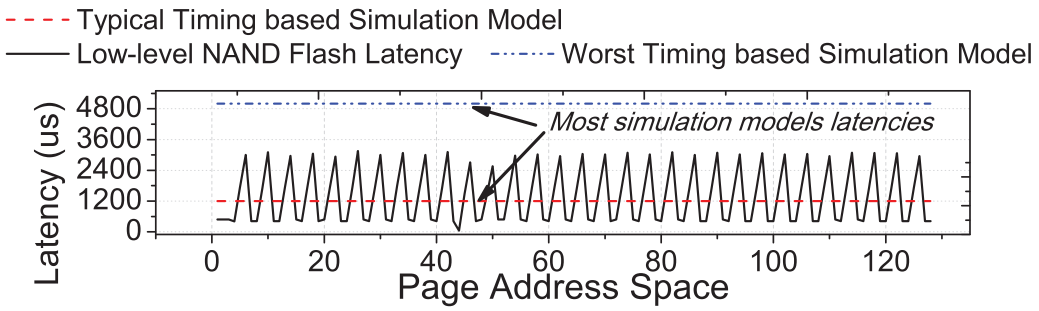
1. INTRODUCTION

While processors have enjoyed doubled performance every 18 months and main memory performance has increased approximately 7% in the same time frame, nonvolatile storage media have not advanced for nearly a decade [Patterson 2004]. Significant effort has been invested in remedying this disparity, and NAND flash has been positioned at the forefront of this effort. Since NAND flash is hundreds to thousands of times faster than conventional storage media and has a small form factor, it has been employed in the construction of devices, such as Solid State Disks (SSDs), Compact Flash, and Flash Cards. NAND flash density increases two- to fourfold every 2 years [Lee et al. 2008], which in turn decreases its cost and allows for widespread deployment in arenas as diverse as embedded systems and high-performance computing. In addition, by introducing multiple planes and dies, NAND flash memory is expected to continue in this trend, as it enjoys the same ease as scaling multiprocessors currently enjoy. As a result of this proliferation, the performance, energy consumption, and reliability of NAND flash memory are becoming increasingly important [Wei et al. 2011]. Further, this proliferation also results in a diversification of target system configurations, such as additional Flash Translation Layer (FTL) logic positioned atop flash, which is frequently tailored to the demands of various NAND-flash-based applications. However, because NAND flash is very sensitive to a large number of parameters, and some latency parameters fluctuate between best and worst case [Grupp et al. 2009], deciding on an optimal NAND flash memory system configuration is nontrivial. Furthermore, NAND flash memory can have many different parameters depending on the memory system types that are employed, for example, single-level cells (SLCs), multilevel cells (MLCs), diverse node technologies (fabrication processes), page sizes, register management policies, memory densities, and pin configurations. Consequently, this large parameter space and the sensitivity of NAND flash to these parameters results in memory systems that exhibit significantly different behaviors.

Unfortunately, a comparison of the different types of NAND flash memory systems becomes even more difficult when multidie and multiplane architectures are considered [Fisher 2008]. In these architectures, scheduling methods and arbitration [Park et al. 2010] among multiple dies and planes are important factors in determining I/O performance [Fisher 2008]. However, the incorporation of these methods and arbiters results in the complexity of flash firmware and controllers being greatly increased. Although simulation-based prior research [Hu et al. 2011; Kim et al. 2009; Maghraoui et al. 2010; Agrawal et al. 2008] revealed performance tradeoffs at the application level, the main focus of these studies was on the SSD as a whole, rather than the NAND flash memory itself; this difference is depicted in Figure 1(a). Such simulations make several assumptions that to varying extents ignore the fluctuating timing behaviors of the diverse I/O operations supported by state-of-the-art NAND flash memory. These assumptions range from being extremely widespread, where the SSD is modeled as having constant time and energy per I/O request, to more confined but still overly simplified, where dies and planes are modeled but the interactions between various NAND commands and components are still represented with *constants*. This implies that the existing simulation models used in these prior studies are tightly coupled to particular flash firmware and policies—performing the exact same study using slightly different firmware or policy sets, such as different page allocation strategies as discussed in Jung and Kandemir [2012], has the potential to result in widely different



(a) Existing SSD simulations (b) Proposed NAND simulation



(c) Latency comparison

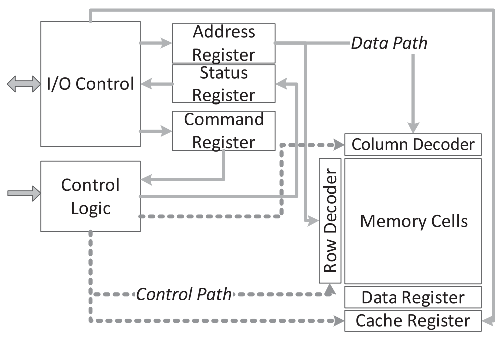
Fig. 1. Concept of a μarch-level NAND flash simulation model (NANDFlashSim). While existing SSD simulators are tightly coupled to flash firmware emulation with a simplified latency model, NANDFlashSim simulates the NAND flash memory system itself with independently synchronous clock domains and detailed NAND operation timing models that are aware of latency variation.

performances and conclusions. (There is a possibility that a specific firmware or policy setting can show a much better performance than others, unlike in the real systems.) Using such imprecise timing models of NAND flash and NAND operations, hardware and system designers may overlook opportunities to improve memory system performance. Furthermore, as shown in Figure 1(c), since these prior studies are oblivious to the *intrinsic latency variation* of NAND flash, they are not able to appropriately model diverse node technologies. In addition, simplified latency models ignore the substantial contributions of flash firmware to the memory system performance. This may result in these designers overlooking research potential regarding new algorithms in/on NAND flash memory systems, such as those involved in internal parallelism handling, wear leveling, garbage collection, the flash translation layer, flash-aware file systems, and flash controllers.

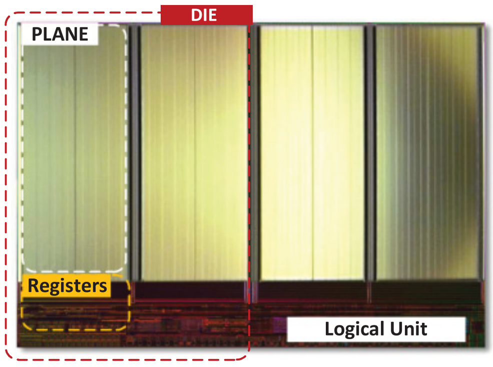
To address these drawbacks, the introduction of a microarchitecture (μarch)-level NAND flash memory system simulation model that is decoupled from specific flash firmware and supports detailed NAND operations with cycle accuracy is required. This low-level simulation model can allow research on the NAND flash memory system itself, as well as on many NAND flash-based devices, as illustrated in Figure 1(b). Specifically, in this article, we propose *NANDFlashSim*, a latency variation-aware, detailed, and highly reconfigurable μarch-level NAND flash memory system based on multidie and multiplane architectures. To the best of our knowledge, NANDFlashSim is the first simulation model to target the NAND flash memory system itself at the μarch level and the first model to provide 16 latency variation-aware NAND flash operations with NAND command set architecture.

The results of our comprehensive experiments using NANDFlashSim showed the following. (1) Most read cases were unable to leverage the highly parallel internal architecture of NAND flash, regardless of the NAND flash operation mode. Specifically, the read throughput improvements between quad dies and octal dies, between four planes and eight planes, and between 4KB and 8KB page sizes are 10.9%, 10.8%, and 10.9%, respectively, while the write throughputs are improved by 91.2% on average. (2) The main contributor to the performance bottleneck is I/O bus activity, not NAND flash activity itself: 50.5% of the total I/O execution cycles is consumed by operations related to such I/O bus activity. The bottleneck is more problematic when *advanced NAND flash commands*, for example, cache and multiplane mode, are applied. (3) MLC NAND flash provides lower I/O bus resource contention than SLC NAND flash, but this resource contention becomes a serious problem as the number of dies increases. (4) A preference for employing many dies rather than many planes provides on average a 54.5% better performance in terms of throughput for disk-friendly workloads [SNIA 2006]. (5) The energy savings are not significant when only I/O bus frequency is increased, whereas the performance is greatly improved. In addition, (6) the recent trend of increasing page size to improve throughput does not guarantee a reduction in energy consumption. This article makes the following main **contributions**:

- *Detailed Timing Model:* NANDFlashSim presents a μarch-level flash simulation model for many NAND flash-based applications. The memory system, controller, and NAND flash memory cells have independent synchronous clock domains. In addition, by employing multistage operations and command chains for each die, NANDFlashSim provides a set of timing details for a large array of NAND flash operation modes, including legacy, cache, internal data move, multiplane, multiplane cache, interleaved die, interleaved-die cache, interleaved-die multiplane, and interleaved-die multiplane cache. These detailed NAND operation modes and their associated timings reveal performance optimization points to NAND flash-based application designers and developers.
- *Intrinsic Latency Variation-Aware Simulation Model:* NAND flash memory, in particular MLC, suffers from intrinsic performance variations when accessing a block. In our observations, the write latency of Micron Technology, Inc [2007] and Hynix, Inc [2009] varies between 250μs and 2,200μs and 440μs and 5,000μs, respectively (maximum latencies are 8.8–11.3 times higher than minimum latencies). Therefore, NANDFlashSim, a cycle-accurate simulation model, is designed to be performance variation aware and employs different page offsets in a physical block. To collect statistics related to the performance variation and validate our simulation model accuracy, we prototyped a NAND flash hardware platform, called the Memory Statistic Information System (MSIS). We present a comprehensive evaluation considering different types of NAND flash and NAND operation on both NANDFlashSim and MSIS.
- *Reconfigurable Microarchitectures:* NANDFlashSim supports highly reconfigurable architectures in terms of multiple dies and planes. This allows a researcher to explore true internal parallelism in such an architecture, because the intrinsic latency variations in NAND flash are exposed. In contrast to prior simulation models, NANDFlashSim removes the dependency on a particular flash firmware, which enables memory system designers and architects to develop and optimize diverse algorithms targeting NAND flash, such as buffer replacement algorithms, wear-leveling algorithms, flash file systems, flash translation layers, and I/O schedulers. Further, to satisfy the needs of the NAND flash-based system researchers, NANDFlashSim can be easily integrated with other system-level simulation models, because it provides streamlined and optional interfaces, which is discussed in detail in Section 6.1.



(a) NAND flash microarchitecture



(b) Die microphotograph [Kim et al. 2011]

Fig. 2. NAND flash memory system internals.

2. NAND FLASH MICROARCHITECTURE

Figure 2(a) illustrates the NAND flash microarchitecture [Micron Technology, Inc 2007] and Figure 2(b) depicts the physical NAND memory cell microphotograph [Kim et al. 2011]. Energy consumption and interface complexity are important factors in NAND flash memory system design. Therefore, interfaces for data, commands, and addresses are multiplexed onto the same I/O bus, which helps reduce pin counts, interface complexity, and energy consumption [Micron Technology, Inc 2007]. Because of this, a host model must first inform the NAND flash package through control logic that it wishes to use the I/O bus before acquiring it. This information is provided via control signals, such as *command latch enable* (CLE) and *address latch enable* (ALE). Similarly, NAND commands are responsible for signaling usage of the I/O bus, in addition to classifying the following NAND operation types.

- *Page / Block.* A page is a set of NAND memory cells and a block is a set of pages (typically 32–256 pages). A physical NAND block makes up a plane.
- *Register.* Registers are used to provide collection and buffering for delayed write-back of small writes and to fill the performance gap between the NAND flash interface and the flash memory cells. Support of multiple registers is a common trend, the purpose of which is to boost NAND flash memory performance. NAND flash is typically composed of a set of cache and data registers.
- *Plane.* A plane is the smallest unit that serves an I/O request in a parallel fashion. In practice, physical planes share one or more word lines for accessing NAND flash cells, which allows the memory system to serve multiple I/O requests simultaneously [Lee et al. 2004].
- *Die.* A die contains an even number of planes and constitutes a NAND flash package. According to the number of dies that are placed into a package, a NAND flash memory is classified as a single die package (SDP), dual die package (DDP), quad die package (QDP), or octal die package (ODP).
- *Logical Unit.* A logical unit consists of multiple dies and is the minimum unit that can independently execute commands and report its status. Multiple dies in a logical unit are interlaced by a chip enable (CE) pin, leading to a reduction in I/O bus complexity and total pin counts.

Since the number of dies that share the I/O bus and CE pins is determined at packaging time, different numbers of logical units are used in DDP, QDP, and ODP. Although state-of-the-art NAND flash provides at most four planes [Kim et al. 2011] and eight dies, our proposed simulation model can be configured to simulate a much larger number of planes and dies in a logical unit.

3. NAND FLASH OPERATIONS

Legacy NAND operations can be classified into three types: *read*, *write* (also referred to as program), and *erase*. While reads and writes operate at a *page* granularity, the erase operation is executed on an entire block. To operate NAND flash memory, first a command is loaded into the command register by raising the CLE signal, which notifies which operation will be executed. Then, a start address for the operations is loaded into an internal address register by raising the ALE signal. When the address has been loaded, the NAND operation can be issued by loading the initiate command. Each of the NAND operations has different timings for data movement. For reads, a page of data is loaded from specific NAND memory cells into the data register. This data movement stage is called *transfer-out of NAND memory cells* (TON). Then, data are sequentially output from the register, byte by byte, which is a process termed *transfer-out of register* (TOR). In the case of a write operation, after the address is loaded, the data can be stored in the data register. This data movement stage, called *transfer-in of register* (TIR), should be processed before loading the initiate NAND command. Following TIR, the NAND flash memory system starts to write data from the register to NAND memory cells, called the *transfer-in of NAND cell* (TIN) stage. In addition to these basic operations, state-of-the-art NAND flash memories support more complex operations to improve system performance [Fisher 2008]. Next, we explain the different I/O modes, which are used in concert with these legacy commands.

3.1. Cache Mode Operation

In *cache mode* operation, data are first transferred to a cache register and then copied to a data register. Then, the NAND flash memory system enters the TIN stage. Meanwhile, the memory system is again available for TIR stage operations using the cache register, because only the data register and memory cells are involved in writing. This cache mode operation overlaps the process of putting data into the register and that of writing data into the NAND memory cells, thereby hiding the TIR time. Exactly like writes, read operations can also take advantage of the cache register. However, in our observations, cache mode operations demonstrate slightly different performances for reads and writes. This is due to the latency-dominating NAND operation, which is further discussed in Section 8.

3.2. Internal Data Move Mode Operation

Applications may need to copy data from a source page to a destination page on the same NAND flash memory. Since data movement from one location to another within flash memory requires external memory space and cycles, a data copy is surprisingly computationally expensive and time consuming. To reduce the time required to copy data, state-of-the-art NAND flash memory supports *internal data move* operations. In these operations, a host is not involved in managing the data copy process. Instead, the host has only to load the source and destination address for copying data into the address registers and commit the internal data move mode NAND command. Then, the NAND flash memory reads data from the source using the data register and directly writes it to its destination, without any data transfer involving the host model. That is, in internal data movement operation mode, data in one page of NAND memory destined for another page can be copied without any external memory cycles. This specialized operation alleviates the overheads of data copying, which notably results in greatly enhanced garbage collection performance [Chang et al. 2004], a critical flash firmware task.

3.3. Multiplane Mode Operation

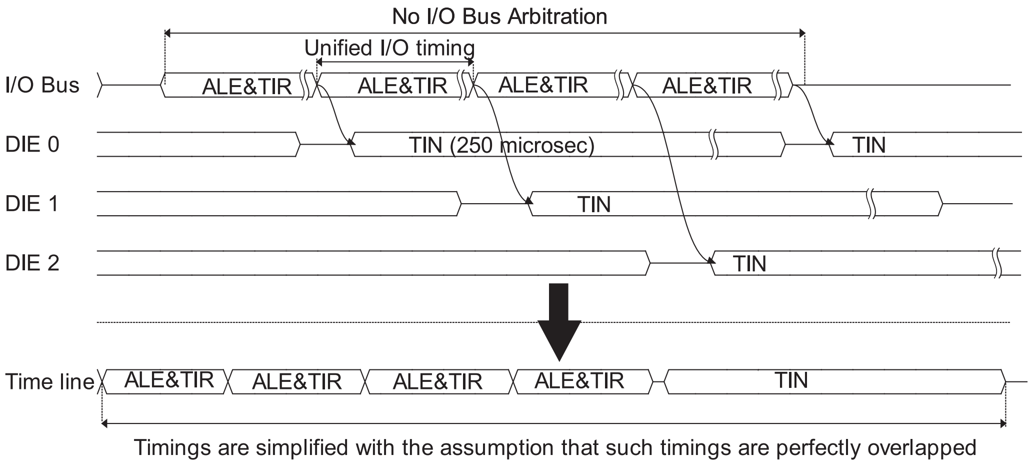
Multiplane mode operations serve I/O requests using several planes simultaneously that are connected by word line(s). Specifically, these operations can enhance performance up to n times, where n is the number of planes in a word line. However, the multiplane architecture leads to limitations for addressing planes. Specifically, in multiplane mode operations, I/O requests should indicate the same page offset in a block and same die address, and should have different plane addresses [Micron Technology, Inc 2007; ONFI Working Group 2011]. These constraints are collectively referred to as the *plane addressing rule*. Therefore, performance enhancement using a multiplane architecture may be limited based on user access patterns. We discuss this issue in Section 8.5. These plane addressing rules must be observed to obtain realistic performance measures of multiplane mode operations. Using these rules, NANDFlashSim provides an accurate implementation of multiplane mode operations, which may be used in any combination with other NAND flash operations.

3.4. Interleaved Die Mode Operation

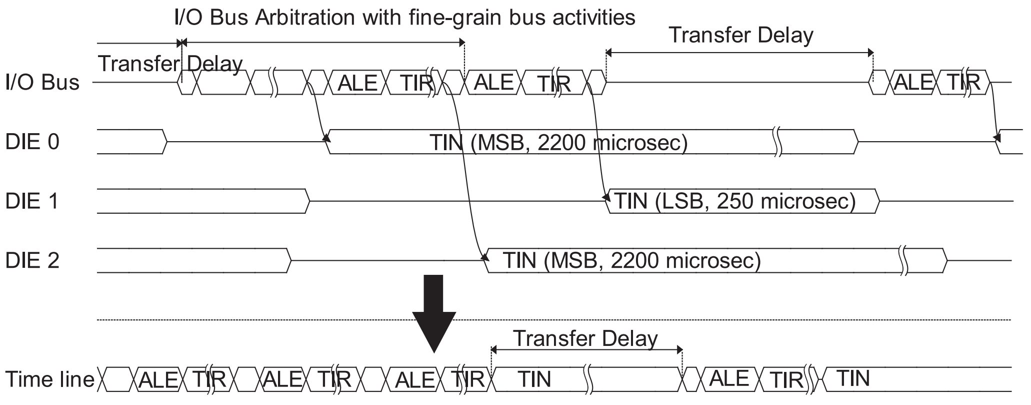
State-of-the-art flash memory shares between one and four I/O buses among multiple dies in order to reduce the number of pins. While sharing the I/O bus reduces energy consumption and complexity, the I/O bandwidth of the system is also reduced. This is because all NAND operations, except those related to NAND memory cells (e.g., TON, TIN), should obtain permission to use the I/O bus before they start executing. Thus, efficient bus arbitration and NAND command scheduling policies are critical determinants of memory system performance. *Interleaved die mode* operations provide a means of sharing the I/O bus and take advantage of internal parallelism by interleaving NAND operations among multiple dies. Unlike multiplane mode operations, interleaved-die mode operations have no addressing restrictions.

It should be noted that all the NAND operations discussed previously can be used in any combination with interleaved-die operations. For example, a host model can issue an interleaved-die multiplane mode operation, which stripes a set of multiplane mode operations across multiple dies. Similarly, interleaved-die multi-plane cache mode operations are possible, which are operations that have the properties of operating in cache mode, being striped over multiple dies and applied to multiple planes. A simplified and approximated latency circulation model with constant times is unable to capture the behavior of and interactions between these different types of operation. Furthermore, intrinsic latency variations exhibited by the NAND flash make it difficult for a latency model with constant timings to mimic elaborate bus arbitration or scheduling NAND commands.

Consider the comparison, shown in Figure 3, between the existing simulation model (with constant time) and variation-aware NANDFlashSim. In the figure, four I/O requests are striped over three dies with interleaved-die legacy write mode. Existing simulation models calculate the latency under the assumption that *timings are perfectly overlapped and interleaved*. Let t_{io} denote the execution time for I/O activities and t_{prog} denote the programming (write) time. Suppose that n_{io} denotes the number of the write requests and $t^{interleaved}_{resp\_legacy}$ denotes the response time for n_{io} requests. In existing simulation models, $t^{interleaved}_{resp\_legacy}$ is simply calculated by $n_{io} * t_{io} + t_{prog}$, as shown in the timeline of Figure 3(a). However, in practice, $t^{interleaved}_{resp\_legacy}$ varies significantly based on system configurations. This is because t_{prog} fluctuates based on the access address, and the transfer delay time is also varied by the service order. In contrast, NANDFlashSim is aware of latency variation and provides a method for scheduling NAND commands and activities with fine granularity.



(a) Typical case timing parameter-based simulation model

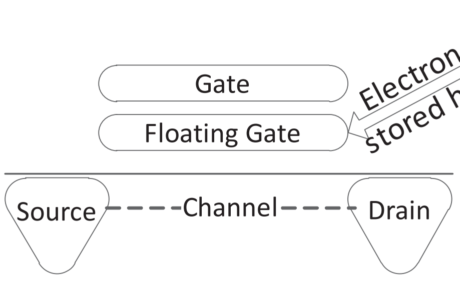


(b) Latency-aware NANDFlashSim

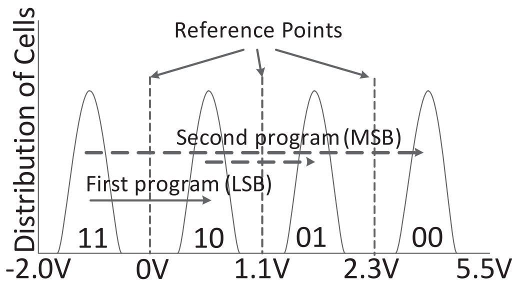
Fig. 3. Timing diagram of interleaved die mode with four legacy writes. While existing simulation models simplify bus activities and assume that latencies are perfectly overlapped and interleaved with constant time, NANDFlashSim employs fine-grained bus activities and therefore is aware of intrinsic latency variations.

4. INTRINSIC LATENCY VARIATION OF NAND FLASH

NAND flash memory has an interesting characteristic, *intrinsic latency variation,* discussed in Grupp et al. [2009] and Lee et al. [2004, 2009], which is that the latencies of the NAND flash memory system fluctuate significantly depending on the address of the pages in a block. Typically, this variation is not specified in the data sheets of NAND flash memory. NAND flash memory puts electrons, which represent cell states, into a NAND flash floating gate. To achieve this, it selects the NAND flash memory cells and makes an electron channel between a source and a drain (see Figure 4(a)). When the channel has been built and voltage over a certain threshold has been applied, electrons can be moved from the channel to the floating gate. This process is called *Fowler-Nordheim tunneling* (*FN-tunneling*) [Lee et al. 2004], which is a well-known programming (write) operation. As illustrated in Figure 4(b), based on differing cell distributions, an MLC NAND flash memory system can identify bit states, such



(a) NAND transistor cell



(b) Example of cell distribution

Fig. 4. NAND flash memory cell organization. MLC NAND flash memory has multiple states in a cell, which causes intrinsic latency variation [Lee et al. 2004].

as “11,” “10,” “00,” and “01,” in a cell.[1] According to the specific bit states for programming, therefore, an MLC NAND flash memory system eventually consumes different amounts of time and power. Specifically, MLC NAND flash is able to store multiple bits on a cell using *incremental step pulse programming* (*ISPP)* [Lee et al. 2004, 2009].

For example, in the first phase, MLC NAND flash programs a cell from the “11” to the “10” or “11” state. This phase represents the least significant bit (LSB) of an MLC cell. In the second phase, the NAND flash reprograms the cell from the “11” or “10” state to a “01”/“11” or “00”/“10” state, respectively, so that the memory cell represents the most significant bit (MSB). Since MLC devices utilize four states using this ISPP, FN tunneling requires more energy and takes a longer time for MSB pages than for LSB pages [Grupp et al. 2009; Lee et al. 2009; Roohparvar 2007]. Because of these NAND flash memory characteristics, one may observe performance variations between worst- and typical-case programming time parameters. Since there is no need for the ISPP for a specific cell in SLC flash, this latency variation characteristic is more pronounced in MLC NAND flash memory.

5. RELATED WORK

There are several prior studies on simulating a NAND flash-based SSD. The SSD add-on [Agrawal et al. 2008] to DiskSim [Bucy et al. 2008] is a popular simulator that models idealized flash firmware. FlashSim [Kim et al. 2009] is another simulator, implemented using object-oriented code for programmatic ease and extensibility. This simulator supports several types of flash software algorithms. Whereas these simulation models compute performance by calculating latency for each of the basic NAND operations, SSDSim [Hu et al. 2011] accommodates latency calculation models for cache, multiplane, and interleaved-die operations of SLC devices at the application level.

Although these simulation models can enable researchers to explore the design trade-offs of SSDs, they have limitations in simulating the μarch-level NAND flash memory, since they greatly simplify NAND flash characteristics, latencies, and energies from a flash firmware perspective. In addition, *these studies are appropriate for only SLC NAND flash memories*. The critical limitations of the previous models are as follows.

- **Simplistic latency assumptions.** These existing simulation models are not aware of NAND flash memory’s latency variations; they implement the flash memory system based on constant times and energies of worst- or typical-case parameters. However, as mentioned in Section 3, the state-of-the-art memory systems are very complex and support diverse NAND I/O operations for high-performance I/O, which

[1]The “0” bit in a NAND flash cell represents the programmed (written) state.

results in the latency varying extremely widely, even between executions of operations of the same type. In contrast, our proposed NANDFlashSim is aware of the latency variations based on MSB and LSB page addresses in a block and provides legacy mode operations, as well as a number of state-of-the-art modes for more complex operations at the μarch level. As a consequence, NANDFlashSim is able to simulate both *MLC and SLC NAND flash*.
- **Coarse-grained NAND command handling.** Moreover, these past studies mimic multidie and multiplane architectures using coarse-grained I/O operations, which means that a series of fine-grained NAND commands to execute an I/O request is not minutely simulated, but instead, is simply modeled with a single I/O operation. Although these studies considered basic I/O timing based on time parameter statistics and internal parallelism of NAND flash memory, the evaluation of accurate memory system latencies is nontrivial. Using multistage and command chains for each of the NAND flash operations, our proposed NANDFlashSim, reconfigurable for multidie and multiplane architectures, provides detailed timing models for NAND operations and manages bus arbitration based on different latencies at the μarch level.
- **Weak model of NAND flash memory constraints.** The memory system's performance and energy consumption can exhibit a variety of patterns because of NAND flash memory constraints. For example, as mentioned in Section 3, multiplane I/O operations should satisfy the plane addressing rules. This constraint results in different performance characteristics according to I/O patterns. Although the past studies considered these kinds of constraints, their simulation was *tightly coupled with specific firmware*. This problem makes it very difficult to explore new memory systems that can be built using NAND flash memory. As opposed to these prior efforts, our NANDFlashSim regulates NAND flash memory constraints at the μarch level and is not tied to any specific flash firmware, algorithm, or NAND flash applications, such as SSDs.

6. HIGH-LEVEL VIEW OF NANDFLASHSIM

NANDFlashSim employs a highly reconfigurable and detailed timing model for various state-of-the-art NAND flash memory systems. NANDFlashSim removes the specific flash firmware and algorithms from the NAND flash simulation model so that memory system designers and architects can employ NAND flash memory systems for various NAND flash-based applications and research, and develop flash software for their specific purposes. To achieve design goals, instead of employing simplified latency calculation models, NANDFlashSim uses a NAND command set architecture and individual state machines associated with the command sets, which results in independent synchronous clock domains. By exposing the details of NAND flash, these mechanisms enable designers and architects to closely study NAND flash performance and optimization points at a cycle level.

6.1. Software Architecture

Figure 5 illustrates the software architecture of our proposed simulation model. NANDFlashSim is composed of a logical unit, a NAND flash I/O bus model, several registers, a controller module, die modules, plane modules, and virtual NAND blocks. A host model can issue any type of NAND flash operation through the NAND I/O bus when the memory system is not busy. NANDFlashSim provides two interfaces to manage NAND flash memory. The first is a *low-level command interface*, which is compliant with the Open NAND Flash Interface (ONFI) [ONFI Working Group 2011]. In this case, the host model fully handles the set of NAND commands for addressing, moving data, and operating NAND flash memory cells. Since a wrong command or inappropriate NAND command sequence can cause the NAND memory system to malfunction, NANDFlashSim verifies the correctness of commands by checking the command/address registers and its

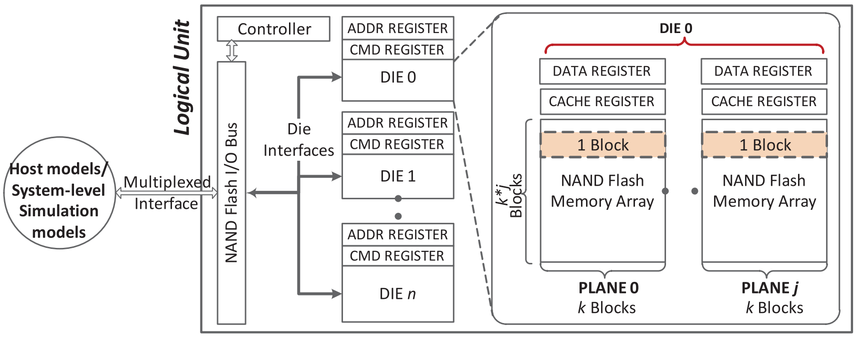


Fig. 5. NANDFlashSim architecture. NANDFlashSim is a reconfigurable μ-level multiplane and multidie architecture. The number of registers, blocks, planes, and dies can be reorganized, and each entity has an independent synchronized clock domain.

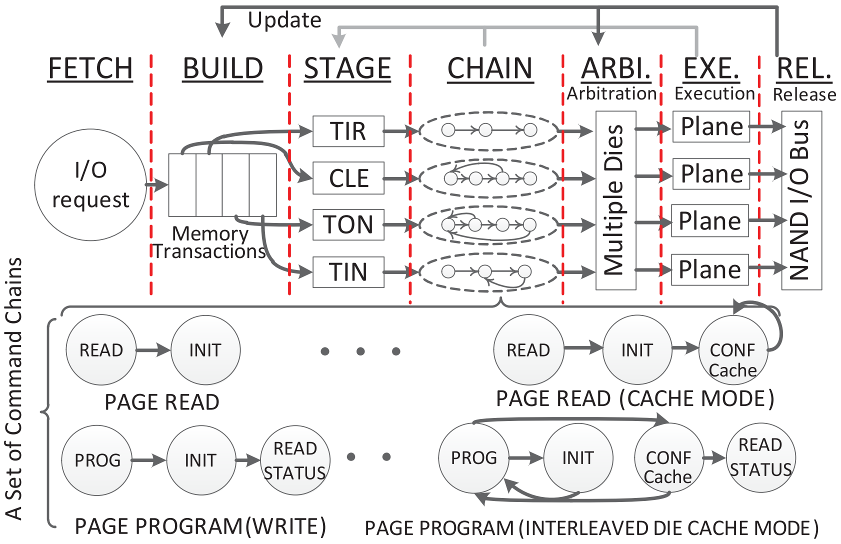


Fig. 6. Process of NAND flash memory transactions and examples of NAND command chains. NANDFlash-Sim handles fine-grained NAND transactions via a NAND command set architecture.

own state machines every cycle. If it detects an incorrect command sequence, it enforces a system fail and notifies the host model. The host model is able to identify the type of failure that occurred by using read-status commands or return codes. The second is a *memory transaction-based interface*. In this case, the host model is not required to manage the set of NAND commands, command sequences, or data movement. Figure 6 visualizes the manner in which NANDFlashSim supports such interface logic. When the logical unit of NANDFlashSim receives a request from the host model, it creates a *memory transaction*, discussed in the next subsection, which is a data structure that includes the command, address, and data. It then places the memory transaction into the internal NAND I/O bus. When the controller module detects a memory transaction on the NAND flash I/O bus, it starts to handle the command sequence based on the command chain associated with the memory transaction. It should be noted that this is handled by NANDFlashSim instead of by the host model. Meanwhile, the logical unit arbitrates NAND flash internal resources (e.g., the NAND I/O buses) and also manages I/O requests across multiple dies and planes. The set of NAND commands generated by the command chain handles the command/address latch and data movement processes, such as TOR, TIR, TON, and TIN, called *stages*. (This is discussed later.)

It should be noted that, using these two interfaces, other simulator models can be easily integrated into NANDFlashSim. Since NAND flash memories are normally integrated with high-level systems, the streamlined and optional interfaces are useful for the users of NANDFlashSim. For example, the flash firmware researchers who develop and optimize wear-leveling algorithms, flash translation layers, I/O schedulers, and so on are allowed to build up their research environments by easily integrating their logics/models with NANDFlashSim modules. The scope of this article is limited to discussions of the NAND flash memory model, as indicated in the title. The extension of NANDFlashSim and integration research remains for future study.

6.2. Clock Domains and Transaction Lifetime

Our simulation model assumes that the logical unit, controller, die, and plane form a module working as an independently clocked synchronous state machine. Many such state machines can be executed on separate clock domains. In general, there are two separate clock domains: (1) the host clock domain and (2) the NAND flash memory system's clock domain. The entities of NANDFlashSim are updated at every clock cycle, and the transaction lives until either it receives an I/O completion notification or NAND flash memory requires a system reset due to an I/O failure. Since the time for a NAND operation can vary from a few to a million cycles, updating all components (e.g., planes, dies, and I/O bus) of NANDFlashSim using the default update interface at every clock can be expensive and ineffective. Therefore, in addition to the default update interface, NANDFlashSim also supports a mechanism to skip cycles, the simulation of which would be of no value. In this mechanism, NANDFlashSim examines all the modules in the logical unit and then finds the minimum clock cycles to reach the next state among them at a given point. NANDFlashSim updates system clocks for its own components based on the detected minimum clock cycles, thereby skipping meaningless cycles in the update process.

7. IMPLEMENTATION DETAILS

7.1. NAND Command Set Architecture

The number of combinations of operations possible with a state-of-the-art NAND flash memory is as high as 16, and each combination has varying timing behaviors. Therefore, NANDFlashSim divides a NAND I/O request into several multiple NAND command sets based on the information specified by the ONFI and updates them at every cycle, as a default. To appropriately operate the NAND flash memory system, this NAND command set architecture is managed by *multistage operations* and *command chains*, as described next.

Multistage Operations. Stages are defined by the common operations that NAND flash has to serve. Specifically, all types of μarch-level NAND operations should have at least one stage, which are classified as CLE, ALE, TIR, TOR, TIN, TON, and BER. CLE is a stage in which a command is latched into the flash memory chip, and ALE is a stage in which an address is loaded into an address register, which is triggered by address latch enable. BER is a stage for erasing block(s). Other stage names that NANDFlashSim employs are the same as those described earlier in Section 2. These stages compose an independently clocked synchronous state machine, as illustrated in Figure 7. This state machine describes different stages for each NAND I/O operation, as visualized in the lower part of Figure 6. All dies have these state machines based on the stage and regulate/validate the correctness of NAND commands and multistage sequences.

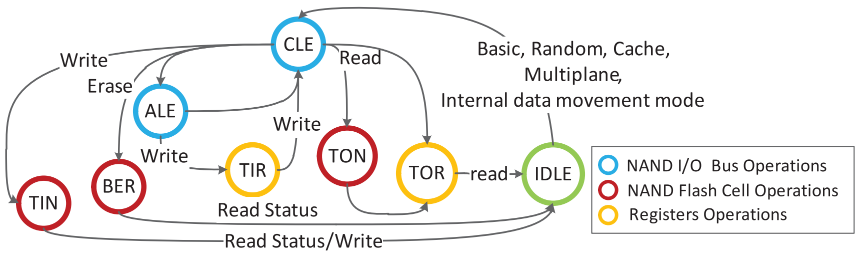
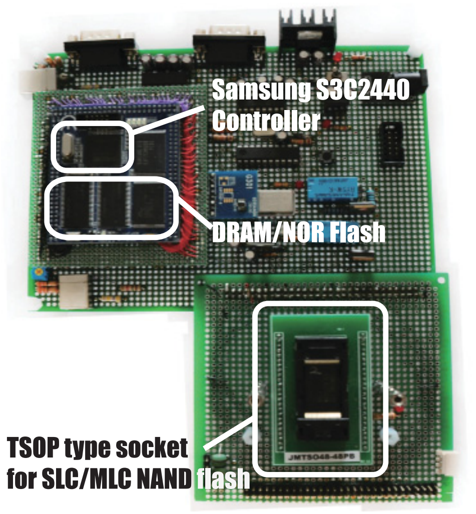


Fig. 7. State machine for multiple NAND stages. Each state is identified by a different type of stage and the states of the machine are transited by different types of NAND command. Since each die has its own state machine, NANDFlashSim provides an independent clock cycle domain per die.

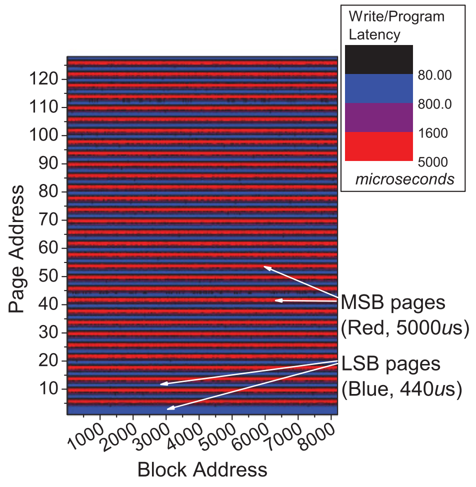
Command Chains. A command chain is a series of NAND commands, and each combination of NAND operations has its own command chain. Although the state machine with a multistage is capable of handling diverse depths of NAND command sets, the introduction of a command chain is required, because many operations have different command requirements and sequences. In addition, the process of transitioning from one stage to another varies based on the command loaded into the command register. For example, as illustrated in Figure 6, the sequence for data movement and the number of commands of the write operation differ from those of the erase and read operations. When a combination of NAND operations with cache, multiplane, or die-interleaved mode is applied, the differences are more obvious. Therefore, NANDFlashSim employs command chains, which are updated by the NANDFlashSim controller and logical unit. In addition, the command chains are used to verify whether the host model manages NAND operation using a correct set of commands and command sequences or not. As described in Section 6.1, if NANDFlashSim with *low-level command interface* detects an incorrect command sequence at runtime, it enforces a system failure and notifies the host. Using multistage operations and command chains, NANDFlashSim defines a NAND command set architecture and provides a cycle-accurate NAND flash model.

7.2. Awareness of Latency Variation

NANDFlashSim is designed to be aware of intrinsic latency variations when it simulates MLC NAND flash. To extract a real performance from and introduce variation characteristics into NANDFlashSim, we implemented a hardware prototype called MSIS, which stands for *Memory Statistics Information System*. MSIS is able to evaluate various types of NAND flash based on different memory sockets, as illustrated in Figure 8. Using MSIS with various NAND flash devices, we commonly found that there are two different types of page in a block, fast pages (LSB pages) and slow pages (MSB pages), according to the write latency on the page. Based on these two sets of pages that are extracted from MSIS, NANDFlashSim can generate different programming timings for different pages. To exactly simulate the latency variation of these pages, NANDFlashSim needs to know the organization of the LSB pages and MSB pages (laid out) within an MLC block. Although the organization of the two sets of pages may vary according to the NAND flash manufacturer, we found that the page organization can be classified into two groups for diverse NAND flash devices (we tested eight devices from four manufacturers, and their technology nodes range from 24 nanometer to 32 nanometer). In the majority of devices, the first four pages are fast, the last four pages are slow, and every pair of pages midblock alternates between fast and slow. In contrast, a block in the other group starts with two fast pages, ends with two slow pages, and places single pages in an alternating fashion between them. These different page organizations are well illustrated in Grupp et al. [2013], where a management scheme


Samsung S3C2440 Controller
DRAM/NOR Flash
TSOP type socket for SLC/MLC NAND flash

(a) Our hardware prototype (MSIS)



(b) Contour map of latency variation

Fig. 8. Implemented evaluation hardware prototype (MSIS). MSIS is used to extract the LSB and MSB page address and to evaluate performance for NANDFlashSim validation.

was proposed to consider the write performance variation across the underlying pages. It should be noted that NAND flash parameters related to these types of pages affect only NAND flash activity, in particular, the transfer-in of NAND (TIN) stage. I/O bus activities, such as CLE, ALE, TIR, and TOR, are not affected by such sets of pages.

7.3. Enforcing Reliability Parameters

NANDFlashSim enforces three constraints to guarantee reliability: (1) the *Number-of-Program* (*NOP*) constraint, (2) *In-order update* in a block, and (3) *endurance*. The NOP constraint refers to the total number of contiguous programmings that the NAND flash memory allows for a block before an erase is required. The plane model in NANDFlash-Sim records the number of programs for each page. If a request attempts to program a page above the NOP limit, NANDFlashSim informs the host model of this violation. In addition, the plane model maintains the page address that was most recently written. When a host model makes a request to program a page that has a lower address than the most recently written page address, NANDFlashSim reports this to the host as a violation of the in-order update constraint. To enforce the endurance limitation, each block in the plane model tracks erase counts. When NANDFlashSim detects a request that would erase a block such that the number of erases would exceed that which the memory system guarantees, it informs the host model of this endurance violation.

Unfortunately, we found that many of the proposed FTL logics atop flash memories violate these important constraints, which can introduce severe reliability problems (beyond the error correction capability) into the real devices. They sometimes overlook information related to the reliability, such as the number of programs and the erase counts (collected from individual flash memory internal data) by focusing their policies or techniques (applied by FTL on top of flash memories) only on system performance improvement. Therefore, in our opinion, the reliability model of NANDFlashSim can verify the correctness of new techniques proposed by flash-based system designers. In addition, these reliability models can provide an environment in which system designers and architects can study NAND flash reliability and explore future research

Table I. NAND Flash Device Characterization

Device Type	Feature	Value
Single-Level Cell	Page Size (Byte)	2,048
(SLC)	No. of Pages Per Block	64
[Micron Technology, Inc 2007]	No. of Blocks	4,096
	Write Latency (us)	250
	Read Latency (us)	25
	Erase Latency (us)	1,500
Multilevel Cell 1	Page Size (Byte)	2,048
(MLC1)	No. of Pages Per Block	128
[Micron Technology, Inc 2007]	No. of Blocks	8,196
	Write Latency (us)	250–2,200
	Read Latency (us)	50
	Erase Latency (us)	2,500
Multilevel Cell 2	Page Size (Byte)	8,192
(MLC2)	No. of Pages Per Block	256
[Hynix, Inc 2009]	No. of Blocks	8,196
	Write Latency (us)	440–5,000
	Read Latency (us)	200
	Erase Latency (us)	2,500

Table II. Workload Characterization

Workloads	Write (%)	Write Req. Size (KB)	Read Req. Size (KB)
Synthesized Write Intensive (swr)	100	2	-
Synthesized Read Intensive (srd)	0	-	2
MSN File Storage Server (msnfs)	93.9	20.7	47.1
Online Transaction (fin)	84.6	3.7	0.4
Search Engine (web)	0.01	99.1	15.1
Shared Home Directories (usr)	2.6	96.2	52.6
Printing Serving (prn)	14.5	97.1	15.1

directions, such as developing new wear-leveling, garbage collection, and address mapping algorithms.

8. EVALUATION

The evaluations of our flash memory model were twofold: *validation of the model* (i.e., the accuracy of the tool) and *design space exploration using the model* (i.e., the usefulness of the tool). We believe that showing both the usefulness and correctness of our tool, instead of only its accuracy, can vitalize our work. Therefore, after validating our tool using various workloads in terms of the latency and throughput, we visit the diverse functions of the tool and explore the design space of flash memory.

For the validation of NANDFlashSim as compared to other real products, we utilized two different types of MLC NAND flash packages [Micron Technology, Inc 2007], that is, Single Die Package (SDP) and Dual Die Package (DDP), and two MLC devices produced by two different manufacturers [Micron Technology, Inc 2007; Hynix, Inc 2009]. In addition, for evaluating NANDFlashSim, we also used SLC- and MLC-type NAND flash. The main parameters for these devices, such as block, page sizes, and latency, are described in Table I. Unless otherwise stated, we used the parameters of MLC1 as default.

Table II analyzes the workloads that we tested. In addition to a number of disk-friendly traces from actual enterprise applications (msnfs, fin, web, usr, and prn) [SNIA

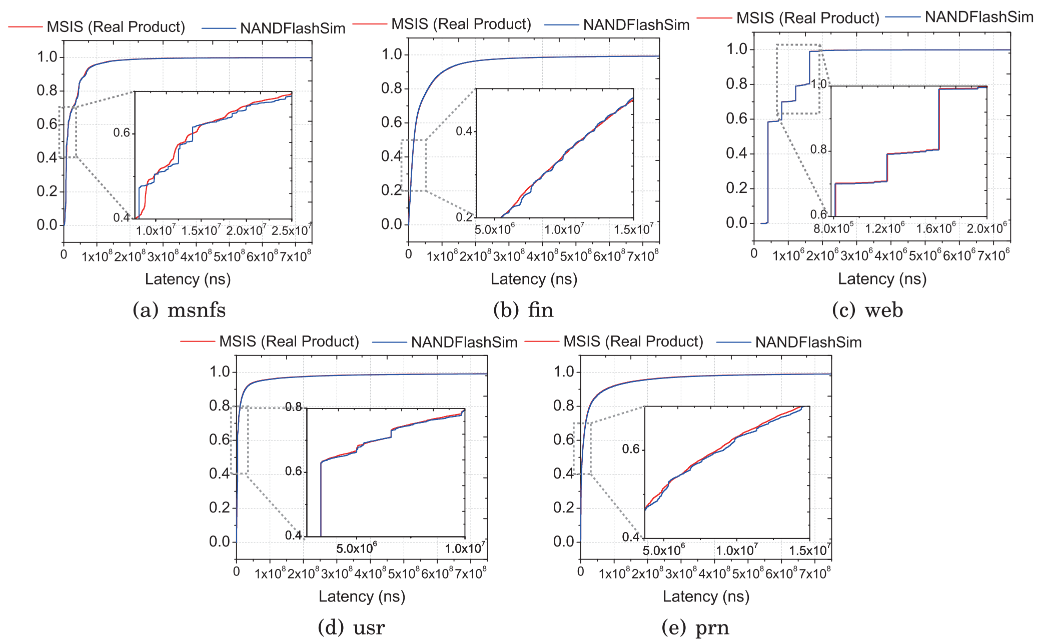


Fig. 9. Cumulative Distribution Function (CDF) of latency. Latencies of NANDFlashSim largely overlap real NAND flash product latencies.

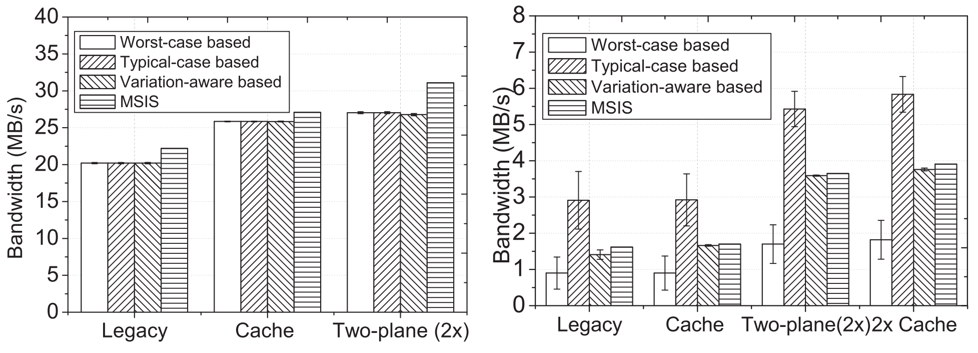
2006], we also synthesized write- and read-intensive workloads, the access pattern of which are fully optimized to NAND flash. Specifically, in the swr and srd workloads, we performed reads and writes of data on different block boundaries and made the access pattern of the workload sequential in the block boundary. With these synthesized flash-friendly workloads, the ideal performance of NAND flash can be evaluated with fewer restrictions. The access patterns of all the workloads tested were used by both the hardware prototype (MSIS) and NANDFlashSim.

We emphasize that *no SSD firmware was involved in this work, because our model addresses NAND flash memories, not SSD*. It should also be noted that we triggered NAND operations in the manner of *plane-level page striping* for all the evaluations.

8.1. Validation of NANDFlashSim

In this section, *we synthetically show the high accuracy of the proposed simulator*. In terms of metrics, the latencies of individual I/O requests and the throughput of the system were measured. In addition, by generating various flash operations, such as legacy, cache mode, multiplane mode, and interleaved die mode, from both the synthesized and real enterprise workloads, we attempted to extensively validate our simulator.

Latency Validation. Figure 9 depicts a cumulative distribution function (CDF) of latency for both NANDFlashSim and MSIS on enterprise application workloads. In this validation, an I/O queue (32 entries) [Intel and Seagate 2003] is employed for handling incoming I/O requests, and legacy-mode, interleaved die-mode, and multiplane-mode NAND commands are issued based on the I/O requests in the queue. It is noted that no SSD firmware is involved, because this is the validation of NAND flash memories, not SSDs. The microscopic illustration of inflections for each CDF is also pictorially embedded. In the figures, the red line represents MSIS latency with MLC2 [Hynix, Inc 2009] and the blue line represents the latency of variation-aware NANDFlashSim. As shown in the figures, the latencies of NANDFlashSim are almost completely overlapped with



(a) Read performance (srd) (b) Write performance (swr)

Fig. 10. Performance comparison on SDP. The performance gap between typical-case/worst-case time parameter-based latency models and the real product is unreasonable.

the real product latencies. Since NANDFlashSim employs a variation-aware timing model by default, it exhibits a performance very close to reality.

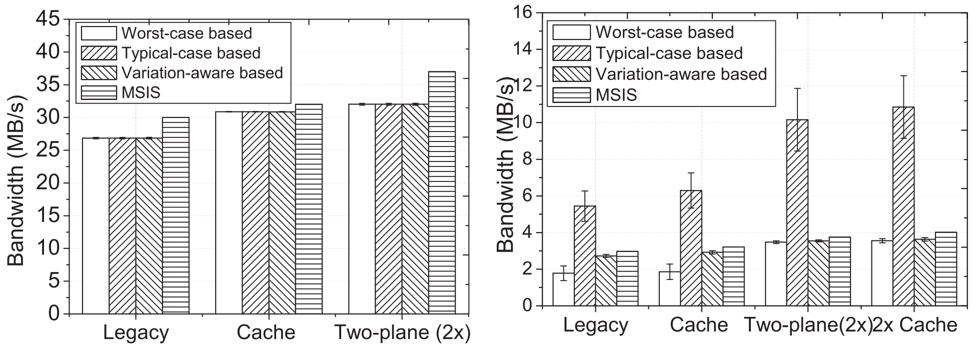
System Performance Validation. In these performance validation tests, we evaluated the performance of our variation-aware-based NANDFlashSim, worst-case timing-based simulation model [Jung and Yoo 2009], typical-case timing-based simulation model [Agrawal et al. 2008; Kim et al. 2009; Hu et al. 2011], and MSIS in terms of bandwidth. In this test, we scheduled NAND I/O commands in plane-first fashion, which means the requests are served with write/read two-plane mode first, rather than striped across multiple dies.

Figure 10 compares the SDP [Micron Technology, Inc 2007] read/write performance of NANDFlashSim and MSIS. The throughput values obtained using NANDFlashSim (with a variance-aware latency model) are close to the performance of MSIS, which can extract the performance of real memory products. In the read cases, NANDFlashSim is no more accurate relative to MSIS than the other timing models, since it does not consider the read performance variation of NAND flash memories.[2] In contrast, write operations show different performances according to the type of latency model employed. Since the write performances vary significantly, there is a performance gap between the performance of MSIS and that of both worst-case-timing and typical-timing parameter-based latency models.

These plots also depict bars of the percentage of deviation between MSIS performances and NANDFlashSim performances among different latency models. Specifically, variation-aware NANDFlashSim provides an approximately 12.9%, 2.1%, 1.6%, and 3.8% deviation in the performance for the legacy, cache mode, 2x mode, and 2x cache mode operation, respectively. This is a significant improvement on the deviation range of 48.7% to 79.6% in typical-case timing parameter-based simulation and from 44.4% to 53.5% in the worst-case timing parameter-based simulation.

Figure 11 illustrates the read/write performance results with DDP [Micron Technology, Inc 2007] on the same test set. The typical-timing parameter-based latency model shows a highly errant performance as compared to MSIS (the deviation range is 83.1% to 170.9%). Although this latency model is the most popular of the SSD simulators, it mimics an ideal performance, which can be achieved *if and only if* the time spent in TIN

[2]The gap between the three simulations and MSIS in the read performance is because there also exist slight variations in flash read performance, although they are not very significant as in write cases. For the read performance, the simulations employed one single timing value, normally the worst-case latency specified in the data sheet. In contrast, a portion of the whole read operations would not require the worst-case latency in the real devices, which were evaluated in our MSIS setting.



(a) Read performance (srd) (b) Write performance (swr)

Fig. 11. Performance comparison on DDP. While the typical-based latency model shows a more discrepant performance than that of multiplane tests, variation-aware NANDFlashSim provides performances close to those of the real product.

is perfectly overlapped with other operation stages and the NAND bus I/O utilization is reasonably high across multiple dies. Although the worst-case parameter-based latency model provides a closer performance (the deviation range is 7.3%–42.0%), it still shows unrealistic performance values. In contrast, the performance deviation range of our current variation-aware NANDFlashSim is between 5.3% and 9.4%. This is because the detailed bus arbitrations across multiple dies, which are based on the intrinsic latency variations, are captured by NANDFlashSim.

In our opinion, the previous validation results show that *NANDFlashSim is a high-fidelity simulation model for various NAND flash operations*. *Based on this high-accuracy simulator*, we now explore the design space of NAND flash memory by handling its major functions.

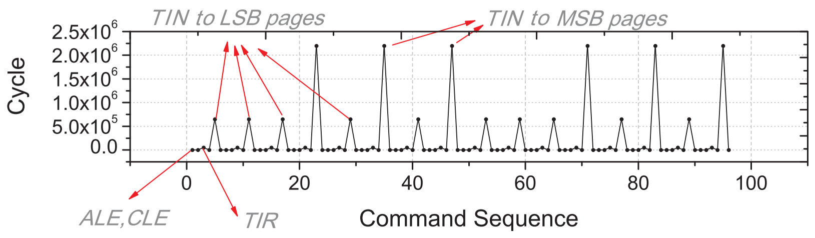
8.2. Individual Cycle Analysis

Figures 12 and 13 illustrate an individual cycle comparison of legacy, cache mode, and 2x mode operations. In this evaluation, we requested eight pages read or write for two blocks, and the size of the requests was 2KB.

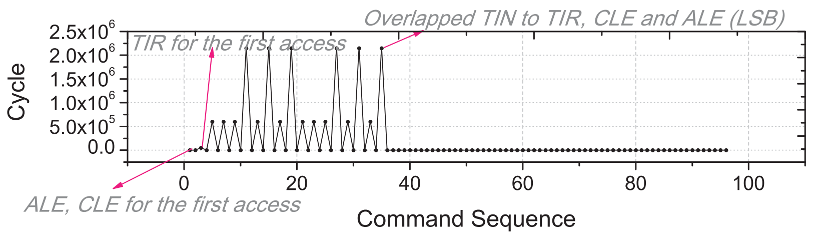
Write Cycles. The cycle analysis of legacy mode writes is illustrated in Figure 12(a). In the write operations, the performances of TINs fluctuate from 650,000 cycles to 5 million cycles. This intrinsic latency variation is one of the reasons NANDFlash-Sim demonstrates a performance closer to reality. Figure 12(b) illustrates the write cycle analysis of cache mode operations. Since latencies for the ALE, CLE, and TIR operations, related to operating the NAND flash I/O bus, can be overlapped with the latency of the TIN operation, the latencies for 16 TIN stages consecutively occur without latencies being spent on operating the I/O bus.

Figure 12(c) depicts the cycle analysis for 2x mode write operations. While cache mode operations save cycles for the I/O bus, the 2x mode operation reduces the number of TIN operations itself by writing data to both planes simultaneously. This is because these planes share one word line. Since the cycles spent in the TIN operation are much longer than the sum of the cycles for ALE, CLE, and TIR operations, throughput is doubled, as shown in Figure 10(b).

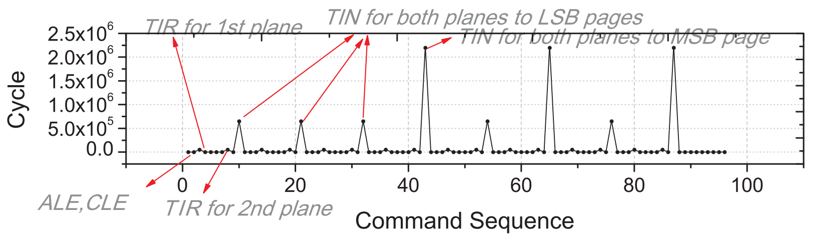
Read Cycles. Figure 13 illustrates the read cycle analysis executed by NANDFlash-Sim. The read operation behaviors for legacy, cache mode, and 2x modes are similar to the write operations, having only two main differences: (1) latencies for the TON operation do not fluctuate as do those of TIN of write operations, and (2) the TOR cycle



(a) Legacy write operation



(b) Cache mode write operation



(c) Two-plane mode (2x) write operation

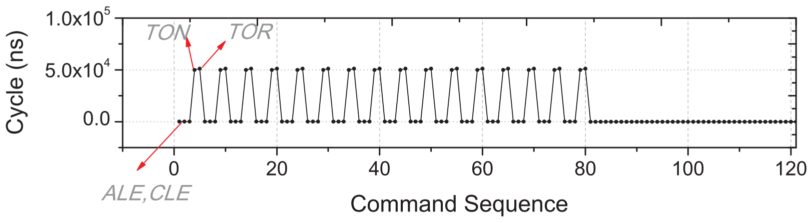
Fig. 12. Cycle analysis for write operations (NANDFlashSim). Note that the command sequence is chronological order based on a set of NAND commands that the host commits, and one cycle takes 1 nanosecond.

fraction of the total execution cycles (see Figure 13(a)) is close to the TON fractions. In Figure 13(b), one can see that the cycles for TOR, which are related to bus operations, are higher in number than those for TON related to operations for NAND memory cells, meaning that reads spend many cycles on the I/O bus operations, which is discussed in more detail in Section 8.4.

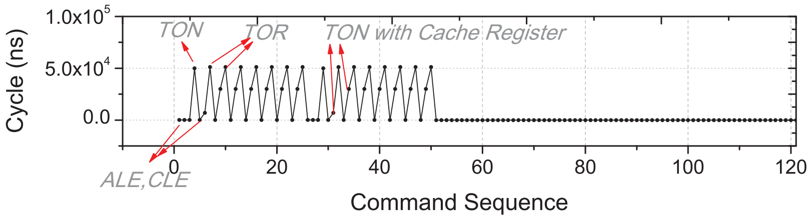
It should be noted that the reason one obtains accurate latency values from NANDFlashSim is that it works at a cycle level and executes NAND operations through multistage operations, which are defined by the NAND command set architecture. In addition, NANDFlashSim reproduces intrinsic latency variations based on different addresses for LSB and MSB pages.

8.3. Performance and Power Consumption Comparison: Page Migration Test

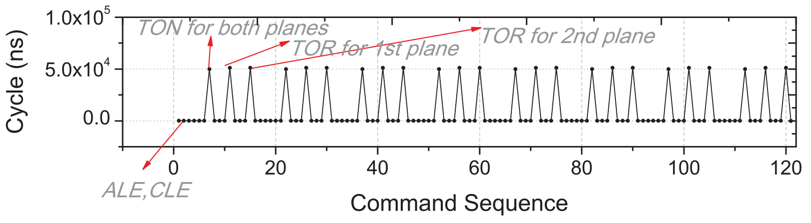
We also evaluated a page migration test, which is a series of tasks consisting of copying pages from source block(s) to destination block(s) and erasing the block(s). This test mimics a very time-consuming component of flash firmware garbage collection. To evaluate the performance of page migration, we read whole pages in the source blocks and wrote them to the destination blocks on NANDFlashSim for various block sizes. In this process, we erased the destination blocks before migrating pages and erased the source blocks after the page migration tasks were complete. These page migration tasks are performed by legacy, cache, internal, 2x, 2x cache, and 2x internal mode



(a) Legacy read operation

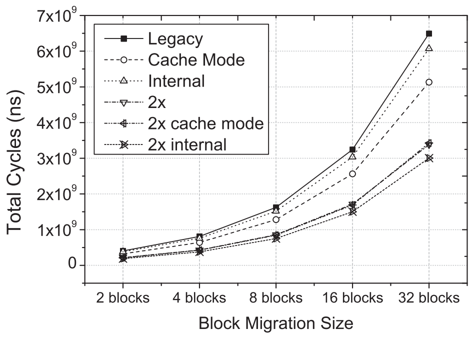


(b) Cache mode read

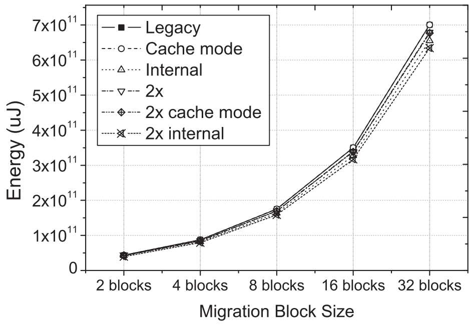


(c) Two-plane mode (2x) read operation

Fig. 13. Cycle analysis for read operations (NANDFlashSim). Since the 2x mode is more complex than other operation modes, it requires more commands to read data. Note that system designers are able to obtain these microscopic cycle analyses for diverse NAND flash operations from the outputs of NANDFlashSim.



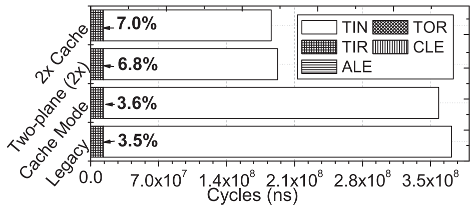
(a) Bandwidth



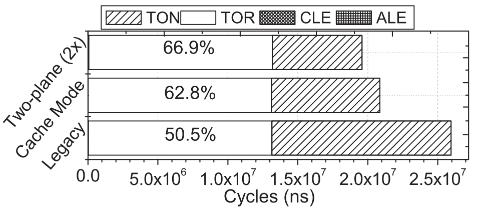
(b) Energy

Fig. 14. Block migration performance comparison and energy consumption. While energy consumption results are similar for different NAND operation modes, 2x cache mode and internal data move mode operations significantly influence the performance enhancement in the page migration test.

operations. As shown in Figure 14(a), there is little performance difference in the page migrations of two blocks, but as the number of blocks in the migration increases, the latencies for the 2x cache mode, internal data move, and 2x internal data move mode operations are about twofold smaller than those of the legacy, cache mode, and 2x



(a) Writes (swr)



(b) Reads (srd)

Fig. 15. Breakdown of cycles. Note that, in reads, TOR operations constitute the performance bottleneck, while TIN operations are on the critical path in writes.

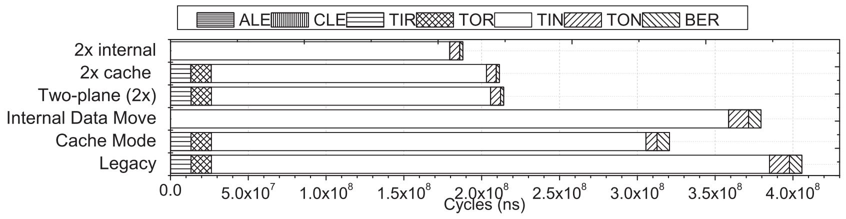


Fig. 16. Cycle analysis for page migration. Internal data move modes remove the most NAND I/O bus activities, thereby improving throughputs.

mode operations. Importantly, the 2x internal data move mode operation outperforms all other operations. In contrast, the difference in the energy consumptions for each NAND I/O operation is not significant, as shown in Figure 14(b). This is because although the source of the latency benefits is internal parallelism, the same amounts of power for operating I/O requests are required by all memory system components.

Figure 16 shows the cycle analysis for each NAND operation type. One can see that internal data move mode operations (including 2x internal) eliminate operations associated with registers (TOR and TIR), thereby improving migration performance.

8.4. Breakdown of Read and Write Cycles

In the write cases shown in Figure 15(a), most cycles are consumed by operations related to NAND flash itself (93%–96.5%). While write operations consume at most 7.0% of the total time performing data movement (TOR/TIN), read operations consume at least 50.5% of the total time doing so. Therefore, although 2x or cache mode is applied to read operations (see Figure 15(b), there is little benefit in terms of bandwidth. In our opinion, this is a reason that considerable research in industry is directed toward enhancing bus bandwidth. However, it should be noted that the write performance cannot be enhanced by any type of high-speed interface, because the speed is dominated by the latency of the TIN stage. It should also be noted that, since NANDFlashSim allows us to count cycles dedicated to each NAND flash stage and command, it helps us determine the operations that constitute the performance bottleneck or the operation that is the best suited for a specific access pattern to improve performance.

8.5. Performance on Multiplane and Multidie Architectures

Multiplane. Figure 17 compares the throughputs observed in NANDFlashSim for varying numbers of planes and transfer sizes. The performance of write operations is significantly enhanced as the number of planes increases, because most of TIN can be executed in parallel. However, for the read operations, these benefits are much smaller than for write operations. This is because cycles for data movement (TOR)

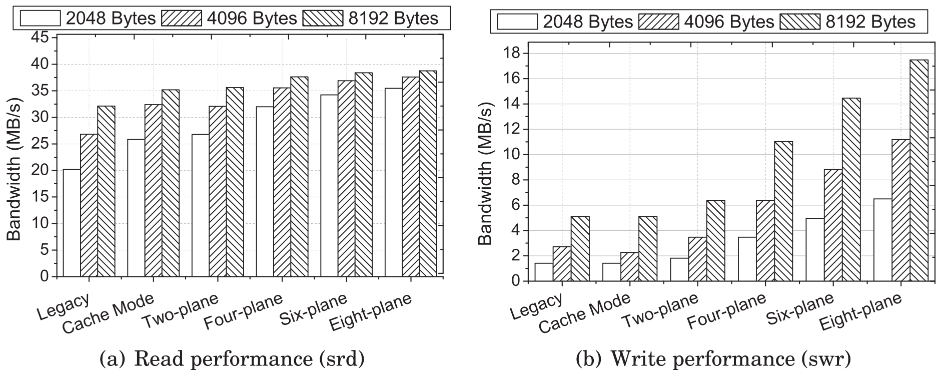


Fig. 17. Multiplane architecture performance with varying page unit sizes. While write throughputs of many-plane architecture are enhanced by 360.9% as compared to single-plane architecture, read throughputs are enhanced by 75.5%.

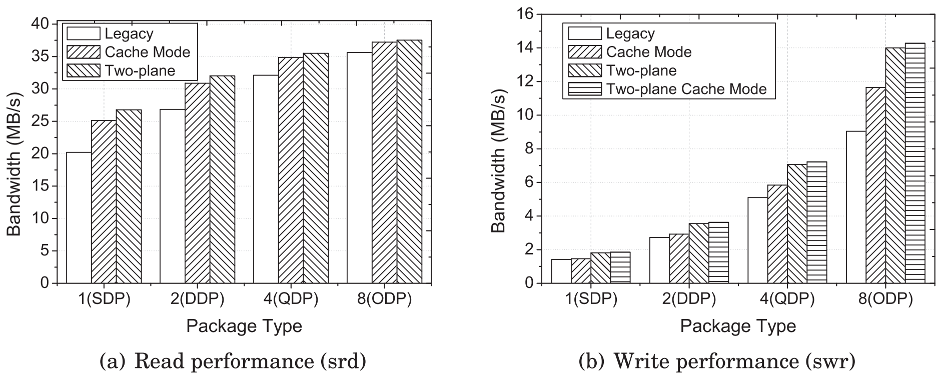
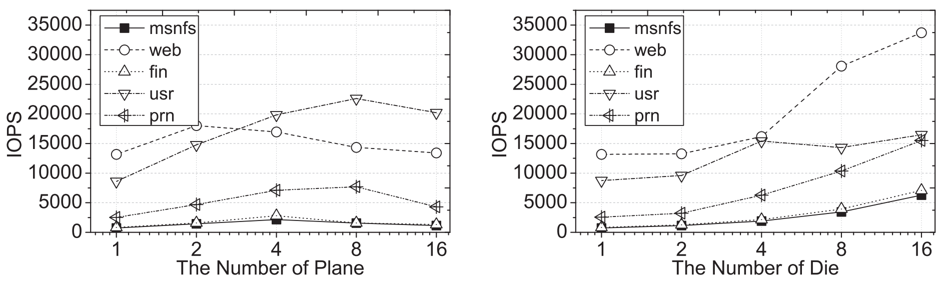


Fig. 18. Multidie architecture performance. While increased die architectures under the swr workload enjoy near-linear enhancement (ODP improves throughput by 541.1% as compared to the SDP ones), it saturates read throughputs with eight dies (76.3% enhancement as compared to the SDP ones).

are a dominant factor in determining read bandwidth and are unaffected by the number of planes. In contrast, for the write operations, where cell programming (TIN) is significantly slower than data movement (TIR), the multiplane mode operation can enhance the write performance. In addition, as shown in Figure 19(a), this performance bottleneck of a many-plane architecture becomes more problematic under disk-friendly workloads. Specifically, the performance gains of the many-plane architecture become limited, starting at a four-plane architecture. The main reason is that most workloads are optimized for traditional blocks without regard for the plane addressing rule. In other words, as the number of planes increases, it becomes difficult to build multiplane mode operations with existing disk-friendly I/O access patterns.

Multidie. Figure 18 illustrates the performance improvement as the number of dies increases. In this section, multiple dies were tied to one NAND flash I/O bus path and they shared one CE pin. Similar to multiplane operations, read performance enhancement (as the number of dies increases) is limited by the latency of the TOR operation. Although multiple dies are able to serve I/O requests in parallel, performance is again bounded by data movement. This is because, during the execution of a TOR operation, the NAND flash I/O bus is not capable of handling another TOR one. Therefore,



(a) Sensitivity to number of planes (b) Sensitivity to number of dies

Fig. 19. Performance sensitivity to the number of planes and dies with actual application workloads. The performance of the many-die architecture is 54.5% better than the performance of the many-plane architecture in terms of IOPs.

regardless of the fact that the 2x or cache mode is applied, the TOR operations constitute the performance bottleneck for reads.

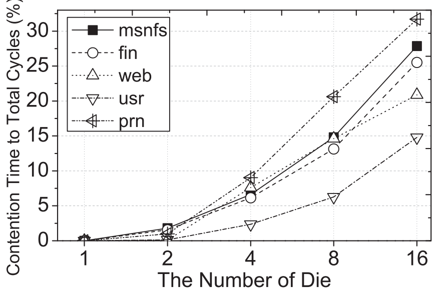
In contrast, as shown in Figure 18(b), the throughputs of write operations are significantly improved by increasing the number of dies. The reason for this benefit is the interleaving of TIN, which is the dominant factor in determining write bandwidths with small bus resource conflicts. It should be noted that NANDFlashSim is able to reproduce/simulate resource (NAND flash I/O bus and dies) conflicts by employing multistage operations and being aware of intrinsic latency variations at the μarch level. As shown in Figure 19(b), unlike many-plane architecture, many-die architecture enjoys performance gains under even disk-friendly real workloads. This is because data can be parallelized across multiple dies with fewer restrictions.

8.6. Performance Sensitivity to Page Size

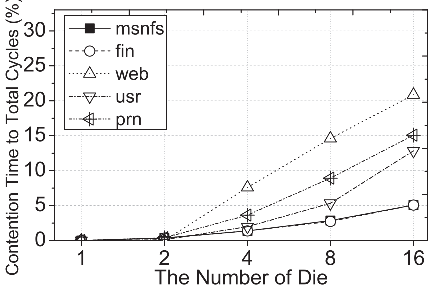
Intuitively, large page sizes can be a good choice for achieving high bandwidth, because many bytes can be programmed or read within the same amount of cycles. However, this intuition is true only for writes. Figure 21 plots the performance sensitivity to different page sizes for diverse read and write operations. While the bandwidth of writes for most operation modes increases as the page size increases, read performances saturate. As explained in Section 8.5, the small enhancements for read operations are due to bus resource conflicts and the large amount of time consumed in data movement.

8.7. Resource Contention

Since multiple dies share the flash interface, I/O bus activities, such as ALE, CLE, TOR, and TIR, should be serialized, which means they cannot be executed simultaneously. Instead, these I/O bus activities can be interleaved across multiple dies at the μarch level. During the interleaving time, I/O requests related to such activities suffer from internal NAND I/O bus resource contention. Figure 20 visualizes the fraction of internal NAND I/O bus resource contention of the total I/O execution time using disk-friendly workloads. As shown in the figure, interleaving I/O bus activities in SLC is 45.2% more competitive than in MLC. The reason is that, since the latencies of MLC activities are much longer than those of SLC activities, there is a greater possibility that a request will be executed with I/O bus activities simultaneously. However, as the number of dies increases, for both SLC and MLC throughput, the fraction of the I/O bus resource contention of the total I/O execution time increases, which is a reason for the performance limitation in the many-dies architecture.

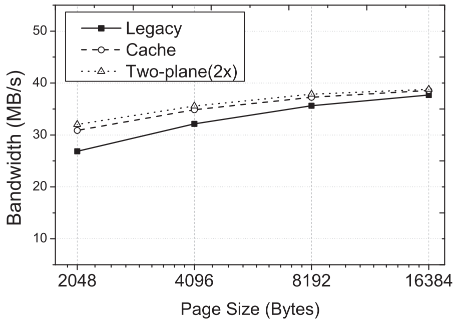


(a) Single-level cell (SLC)

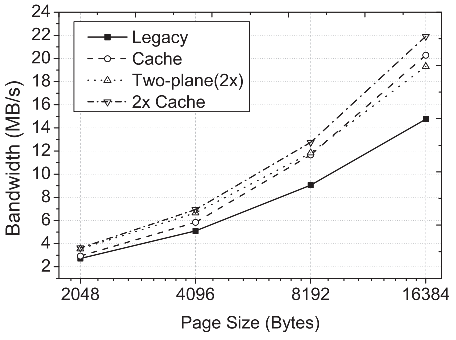


(b) Multilevel cell (MLC)

Fig. 20. Resource contention comparison of SLC NAND and MLC NAND devices. Resource contention influences MLC flash more than SLC NAND flash, but the contention problem remains an issue and becomes more serious as the number of dies increases.



(a) Read (srd)



(b) Write (swr)

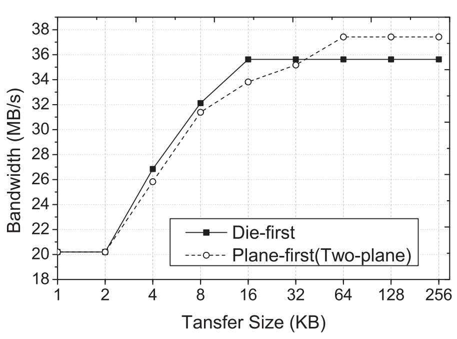
Fig. 21. Sensitivity to page organization (2x, DDP). Most read performances are bounded because of TOR times and NAND flash I/O bus competition.

8.8. Scheduling Strategy

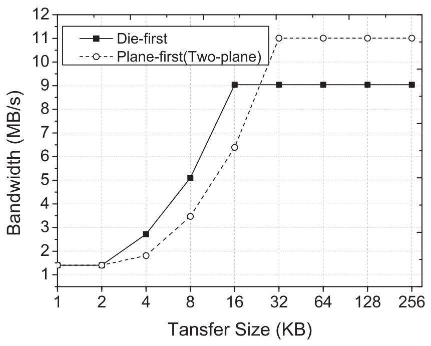
To test potential research directives on NAND command scheduling strategies, we implemented two simple command schedulers in the logical unit of NANDFlashSim: (1) die first and (2) plane first. The die-first scheduler simply stripes I/O requests as they arrive over multiple dies rather than planes. In the plane-first scheduler, I/O requests are collected into two pages upon arrival and served to multiple planes rather than striping them across dies. As illustrated in Figure 22, since multiple dies share one I/O bus, performances saturate faster than with the plane-first scheduler. Although plane-first operation provides a better performance, the die-first scheduler is more flexible in serving I/O requests of a smaller size. This is because multiplane operation performance is limited by plane addressing rules (see Section 3.3), whereas multiple dies can be interleaved to serve I/O requests without any addressing constraints.

9. ENERGY SIMULATION USING NANDFLASHSIM

In addition to the high-fidelity performance simulation, NANDFlashSim can provide an energy simulation by modeling its energy consumption, which is described in detail in this section. Based on the model, we first profile and analyze the energy consumption of NAND flash memory. Further, the effectiveness of varying system components, such as interface and page size, is explored, which provides interesting observations.



(a) Read performance (srd)



(b) Write performance (swr)

Fig. 22. Effects of different NAND command scheduling policies. Based on different transfer sizes and scheduling policies, the performance enhancement with multidie and plane architecture is different.

9.1. Energy Model

In the same manner as it simulates the execution time, NANDFlashSim calculates the flash energy consumption by each I/O operation based on the command chain of the I/O request. By estimating the energy consumption of each stage and summing it, the entire energy consumption for the I/O operation is estimated. Based on the NAND command set architecture described in Section 7, the three legacy operations, read, program, and erase, are given as Equations (1), (2), and (3), respectively:

$$E_{read} = (E_{CLE} + E_{ALE} + E_{CLE}) + E_{TON} + E_{TOR}, \tag{1}$$

$$E_{program} = (E_{CLE} + E_{ALE} + E_{CLE}) + E_{TIR} + E_{TIN} + (E_{CLE} + E_{TOR}), \tag{2}$$

$$E_{erase} = (E_{CLE} + E_{ALE} + E_{CLE}) + E_{BER} + (E_{CLE} + E_{TOR}), \tag{3}$$

where the stages (CLE, ALE, TIR, TOR, TIN, TON, and BER) of the legacy operations are explained in Section 7.1. A series of two CLE stages and one ALE stage that commonly appears in the start of the three operations is for moving control information into the memory. The set of (CLE, ALE, and CLE) corresponds to the op-type command, array address, and initiate command, respectively. For the read operation, data are transferred out of the data register (TOR) after reading them from the memory cell (TON). In contrast, data are transferred into the register (TIR) and further written to the memory cell (TIN) for program operation. In the case of an erase operation, data in the memory cell are removed (BER) without any data movement around the register. Particularly, to check whether the operation on memory cells is successful or not, a read command (CLE) and the movement of status data (TOR) are followed for both the program and erase operation.

When a flash executes an I/O operation, each stage dissipates a different amount of power for a different amount of time. We classify the stages based on the flash-internal hardware components that are activated in each stage.

CLE/ALE. In the CLE stage, a command is loaded into the command register through the external interface, whereas target address information is loaded into the address register in the ALE stage. Since loading a command or the address information into the command or address register is like writing a few bytes of data into a small H/W latch, the CLE/ALE stages consume little power, as well as few execution cycles. Accordingly, NANDFlashSim produces the energy consumption of the CLE/ALE stages by borrowing the power values from LATCH data sheets and multiplying them by the execution cycle value from the timing simulation.

TIR/TOR. To write user data into the flash memory, first, the data are loaded into the internal data register through the flash interface before the memory cells are accessed, which is the TIR stage. In symmetry with TIR, moving user data from the data register to chip external over the flash interface (TOR) to provide it to the user is a necessary stage for read operations. Unlike control data, such as command and address, user data are so big (usually several kilobytes) that the system components involved in the TIR/TOR stage (flash interface and data register) spend a large number of execution cycles and consume considerable energy. Since these stages are quite similar to the write/read operation of DRAM in terms of the medium, interface, and data unit size, we refer to the power values from DRAM data sheets.

TIN/TON/BER. TIN for program, TON for read, and BER for erase are memory-cell-related stages. TIN moves a page of data from the data register to the memory cells, whereas TON transfers it from the memory cells to the data register. Since only the data register and the memory cells are involved in these stages, other major system components, such as flash interface, remain idle. The TIN stage requires a large number of electrons to be moved, which takes a large amount of time and consumes a considerable amount of power. In contrast, the TON stage of read operations takes less execution latency than TIN, since it performs only the sensing of memory cells. Conversely, the BER stage for erase operations triggers a large number of electrons to move out without involving the data register. Since the unit of the NAND flash erase operation is magnitudes larger than the unit for read and program, BER is much more time and energy consuming than the other stages. The majority of the NAND data sheets describe the power characteristics of the TIN, TON, and BER stages.

In a previous study, flash power was modeled [Mohan et al. 2013], and an accurate power estimation based on a huge amount of circuit-level parameters was provided. While this study provides new insights for understanding circuit-level power and energy, NANDFlashSim presents an easily accessible model, which can capture "system-level energy characteristics" by being aware of flash architecture and the cycle variations therein.

9.2. System Energy Consumption

As one of the features of NANDFlashSim, our simulation of the energy consumption of a flash device using various workloads is described in this subsection. We first profile the energy consumption of our real workloads collected from an Android device. Second, the NAND flash energy characteristics are examined by analyzing the energy consumption for each legacy operation.

Experimental Setup. As a flash with abundant system parameters, a representative 8GB Quad Die Package (QDP) SLC device with a 4KB page size [Micron Technology, Inc 2009] was utilized to investigate the energy simulation of NANDFlashSim. In addition to the timing characteristics, a few important electrical characteristics of the target device were additionally specified, as shown in Table III. Further, we collected I/O traces from an Android-based handheld device to simulate the energy consumption of real workloads. The reason we attempted to execute such workloads is that the energy consumption of flash-based storage systems employed in mobile devices is one of the main design concerns. The most likely user scenarios, including playing videos, playing games, taking pictures, taking videos, playing music, and manipulating user files, were conducted on the device. Table IV shows the read/write proportion of each scenario, which categorizes *VideoPlay*, *Game1Play*, *Game2Play*, and *MusicPlay* as the read-intensive workloads and the others as the write-intensive workloads.

Table III. NAND Flash for Energy Simulation

Device Type	Feature	Value
Single-Level Cell	Page Size (Byte)	4,096
Quad Die Package	No. of Pages Per Block	128
[Micron Technology, Inc 2009]	No. of Blocks	16,384
	Write Latency (us)	230–500
	Read Latency (us)	25
	Supply Voltage (V)	3.3
	Array Op. Current (mA)	20
	Flash I/F Frequency (MHz)	40
	Flash I/F Current (mA)	5

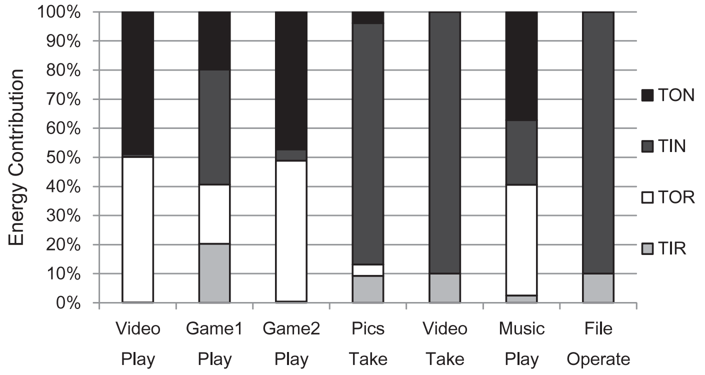
Table IV. Workloads for Energy Simulation

Workloads	Write (%)	Write Req. Size (KB)	Read Req. Size (KB)
Playing videos (VideoPlay)	0.2	7.6	110.6
Playing game1 (Game1Play)	17.8	6.2	82.8
Playing game2 (Game2Play)	0.9	4.3	24.6
Taking pictures (PicsTake)	70	4	21.6
Taking videos (VideoTake)	100	321.4	-
Playing music (MusicPlay)	6.1	5.3	126.9
Manipulating user files (FileOp)	99.9	615	1.3

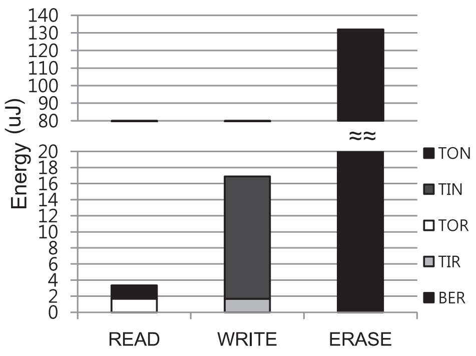
Android Workloads. Figure 23(a) shows the energy contribution of stages when each Android user scenario is executed on NANDFlashSim, mimicking the target device. For read-intensive workloads, such as *VideoPlay* and *Game2Play*, TON and TOR are responsible for most of the total energy consumption. (Recall that TON and TOR are major stages of the read operation, whereas TIN and TIR are representative stages of the write operation.) An interesting finding from the simulation of these workloads that include few write operations is that TON and TOR share the total energy evenly. In contrast, TIN also makes a very significant contribution to the total energy consumption for *Game1Play* and *MusicPlay*, which contain more writes than other read-intensive workloads, but still very few as compared to the number of reads, as described in Table IV. Another observation here is that the energy contribution of writes is comparable to that of reads, although the number of writes scarcely contributes to the total operation count. On the other hand, TIN is the most critical energy consumer for write-intensive workloads, such as *PicsTake*, *VideoTake*, and *FileOp*.

Legacy Operations. To further analyze our observations and the energy consumption characteristics of real workloads described earlier, we profiled the energy consumption of the legacy operations consisting of read, write, and erase operations, which are shown in Figure 23(b). The energy consumption of a page read, page write, and block erase is 3.35, 16.9, and 132.2uJ, respectively. The extreme energy gap among the three legacy operations is caused by the different operational units and latency gaps. Read and write operate at the unit of pages, whereas erase operates at the unit of a block consisting of hundreds of pages. Moreover, there is a very large latency gap between tR (data transfer time from the memory cell array to the data register, or execution time in TON) of read operations and tPROG (data transfer time from the data register to the memory cell array, or execution time in TIN) of write operations.

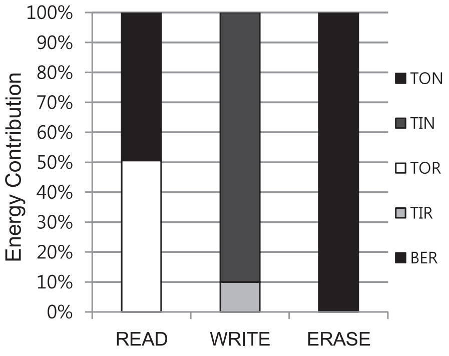
Instead of a direct energy comparison of the three legacy operations, an analysis on the energy contribution of stages for each legacy operation can provide the energy consumption characteristics of NAND flash memory. As can be seen in Figure 23(c), the total energy is consumed in only one or two of the stages, as described in Figure 7.



(a) Energy contribution of stages for Android workloads



(b) Energy consumption of legacy operations



(c) Energy contribution of stages for legacy operations

Fig. 23. Energy contribution of stages for real workloads from an Android device and legacy operations on MLC3.

For the three legacy operations, the common stages, including IDLE, CLE, and ALE, are negligible in terms of the energy consumption. Regarding CLE and ALE, only a few bytes for the command and address information are transferred from the host to the flash chip. This small data movement activates the corresponding system component (H/W registers) for a very short time, which does not contribute to the total energy consumption. In addition, the energy consumption of the IDLE stage can be ignored as well, because the NAND flash dies dissipate little power when they are idle. Note that the number of cycles spent in the IDLE stage is significant, as the remaining three dies of QDP MLC3 are idle while the target die is processing the operation.

The energy consumption of read and write operations is largely divided into two parts: data movement on the flash system interface and NAND memory array operations. (Recall that TOR of read and TIR of write indicate the data movement over the interface, whereas TON of read and TIN of write represent the memory-array-related operation). In contrast, there is no data movement between the external host and the NAND chip in the case of an erase operation, so that the erase operation consists of only a memory array operation BER. For read operations, TOR and TON share equally in the energy consumption, which shows that the system interface is also a critical energy bottleneck, regardless of the NAND medium. However, this does not apply to the write operation. Although the energy consumption of the system interface

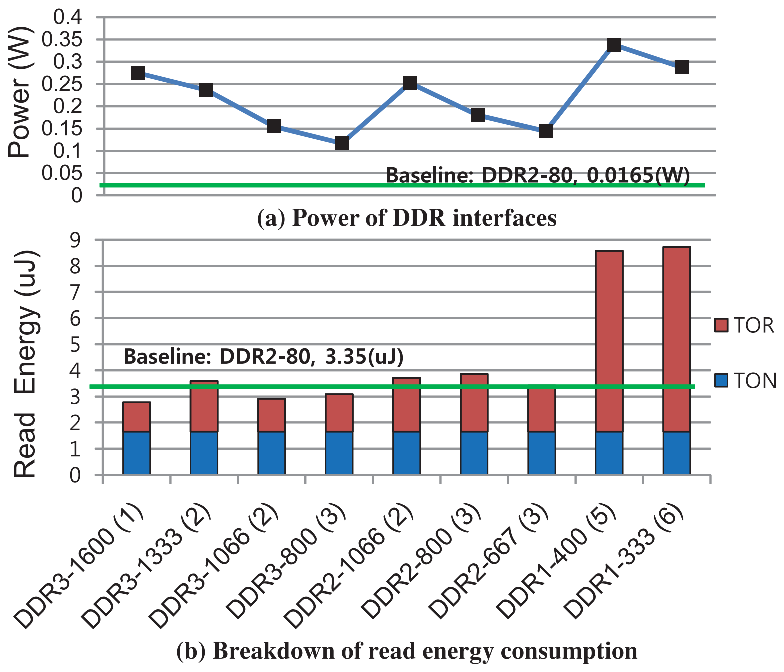


Fig. 24. (a) Operating power and (b) breakdown of the read energy consumption under different DDR interfaces, x-axis: DDR protocol - data rate (tRC).

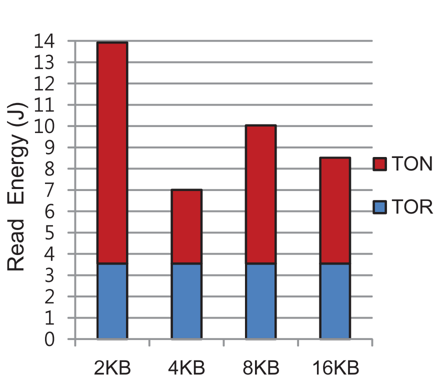
for a write operation is almost the same as that for a read operation (recall that the same amount of 4KB page data moves over the same interface for both operations without direction and refer to the same raw energy values of TOR and TIR in Figure 23(b)), the proportion of it (TIR) to the memory operation (TIN) is only 10% of the total energy consumption. This is because tPROG is much longer than tR. From the breakdown of the energy consumption for legacy operations, one can identify the flash system interface as the major energy-hungry component, particularly for read operations.

9.3. Effectiveness of Varying System Components

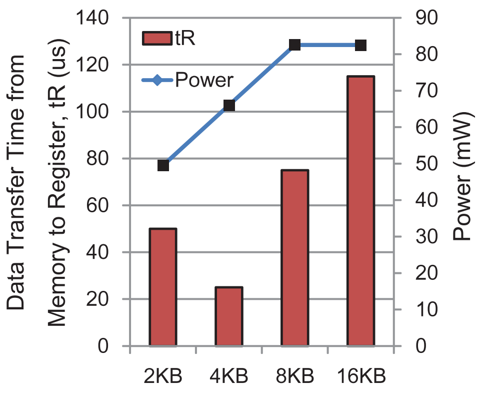
The effectiveness of changing power-hungry system components, such as flash interface and page size, is evaluated here. In addition, we discuss our important findings based on the energy simulation results under different system components.

Varying Interface. As stated in the previous subsection, the system interface is identified as one of the power-hungry system components. Therefore, we here evaluate the effectiveness of changing the flash external interface. According to ONFi 3.x [ONFI Working Group 2011], a NAND flash employs the SDR, DDR, or DDR2 protocol with different frequencies as its external interface. However, the interface specification of the recent version of ONFi is defined as having very low frequencies, such as 100MHz, which is 200MT/s in the case of the DDR interface. To further increase the performance of the interface, we consider the DRAM system interfaces, where the DDR protocol is widely adopted. Unlike the current NAND interfaces, DRAMs employ very high-data-rate DDRx interfaces.

Figure 24(b) illustrates the breakdown of the energy consumption of a 4KB read operation under the different versions of the DDR interface. The x-axis indicates the



(a) Breakdown of read energy consumption



(b) Operating power and tR

Fig. 25. Energy consumption of sequential read operations of 8GB data under different page sizes on MLC3. The latency and power of the TON stage explain the energy consumption patterns.

DDR protocol, data rate (MT/s), and tRC (ns), respectively. (tRC indicates the time to read a unit of data out of the system over the interface.) The TOR energy varies depending on the interface, while the TON energy remains the same, since we change only the flash interface independently of other system components. As compared to the energy consumption (3.35uJ) of the baseline MLC3 with a DDR2-80 interface (tRC = 25ns), the energy gain when more expensive and higher performance interfaces are used is small. In the case of DDR1-333 or -400, the energy consumption becomes significantly worse.

As the data rate (or frequency) of the flash interface increases, the tRC value decreases, which means a page of data can be moved more quickly out of a NAND chip. This is verified by our simulation results, where DDR3-1600 is the best and DDR1-333 is the worst in terms of performance. The data movement latency on the system interface in our baseline MLC3 with DDR2-80 takes over 80% of the total read latency, whereas it is reduced by 15% by employing DDR3-1600. In contrast, as shown in Figure 24(b), there is no large gain in the energy consumption when higher data rate interfaces are adopted. DDR3-1600, DDR3-1066, and DDR-800 save little energy, whereas the others save none. This is because the highly increased power dissipated by the interfaces with high frequencies, shown in Figure 24(a), offsets the improved latency. According to the power values shown in Figure 24(a), as the frequency of the system interface increases, the power dissipation also seems to increase. In addition, the higher the version of a DDR protocol is, the better the DDR protocol is in terms of the power consumption. (DDR3 is more power efficient than DDR2 and DDR1.) Therefore, when designing an energy-efficient NAND flash chip, one should carefully select the target interface by taking power efficiency, performance, cost, and the frequencies of the related components into account. In addition, power optimization of the flash interface should be further studied for the energy optimization of the NAND flash memory.

Varying Page Size. Having addressed the system interface, we now change the page size to evaluate its influence on energy consumption. Figure 25(a) depicts the energy consumption of sequential reads of 8GB data when changing the page size from 2KB to 16KB. The TOR energy remains almost constant, since the total amount of data (8GB) transferred over the selected interface is the same, although the page size varies. This means that the TON energy shapes the entire energy consumption. (The effectiveness for write operations is reflected in similar energy consumption patterns to those of the read operations.)

When using a 4KB page, the TON energy, as well as the total energy consumption, can be minimized. The energy consumption of TON depends on the tR value and the power dissipated during TON, both obtained from NAND data sheets and specified in Figure 25(b). The value tR varies significantly over different page sizes, whereas the operating power does not. (According to multiple vendor data sheets, tR is not proportional to the increase in page size.) This is the reason that no pattern is present in the energy consumption in Figure 25(b). Therefore, we can conclude that the recent trend of increasing page sizes may not lead to significant energy benefits.

10. SIMULATION SPEED AND DOWNLOAD

The current version of NANDFlashSim is capable of executing 824 I/O requests (2KB) per second for DDP and 295 I/O requests per second for ODP with MLC1. The simulator performances were measured on a machine with a virtualized dual core, 1GB memory, and 200GB disk. The source code can be downloaded from http://nfs.camelab.org.

11. CONCLUSION

Since NAND flash memory is sensitive to a large number of parameters and the latency variation in some performance parameters is significant, determining the NAND flash memory configuration that allows optimal performance is nontrivial. A comparison of various NAND flash memory architectures becomes even more difficult when multidie and multiplane architectures, latency variations, energy consumption costs, reliability issues, and addressing restrictions are considered. Therefore, in this article, we proposed NANDFlashSim, a detailed and highly configurable low-level NAND flash simulation model. Its awareness of intrinsic latency variations allows NANDFlashSim to support detailed timing models for 16 I/O operations. Our ongoing work includes incorporating a 400MHz high-speed NAND interface (not yet published) and implementing a multiple logical unit on-chip architecture. In addition, we plan to apply our simulation model to cycle-accurate Green Flash [LBNL LBNL] and the Tensilica Xtensa simulation model [Cadence] of the hardware/software codesign platform for exascale computing [Wehner et al. 2011].

REFERENCES

Nitin Agrawal, Vijayan Prabhakaran, Ted Wobber, John D. Davis, Mark Manasse, and Rina Panigrahy. 2008. Design tradeoffs for SSD performance. In *Proceedings of USENIX ATC*.

John S. Bucy, Jiri Schindler, Steven W. Schlosser, and Gregory R. Ganger. 2008. The disksim simulation environment version 4.0 reference manual. In *Parallel Data Laboratory*.

Cadence. Hardware and Software Development Tools. http://www.tensilica.com/products/hw-sw-dev-tools/.

Li-Pin Chang, Tei-Wei Kuo, and Shi-Wu Lo. 2004. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Transactions on Embedded Computing Systems* 3, 4 (November 2004), 837–863.

Ryan Fisher. 2008. Optimizing NAND flash performance. In *FlashMemory Summit*.

Laura M. Grupp, Adrian M. Caulfield, Joel Coburn, Steven Swanson, Eitan Yaakobi, Paul H. Siegel, and Jack K. Wolf. 2009. Characterizing flash memory: Anomalies, observations,and applications. In *Proceedings of MICRO*.

Laura M. Grupp, John D. Davis, and Steven Swanson. 2013. The harey tortoise: Managing heterogeneous write performance in SSDs. In *Proceedings of the USENIX 2013 Annual Technical Conference*.

Yang Hu, Hong Jiang, Dan Feng, Lei Tian, Hao Luo, and Shuping Zhang. 2011. Performance impact and interplay of SSD parallelism through advanced commands, allocation strategy and data granularity. In *Proceedings of ICS*.

Hynix, Inc. 2009. NAND flash memory MLC datasheet, H27UBG8T2A. http://www.hynix.com/.

Intel and Seagate. 2003. *Serial ATA Native Command Queuing: An Exciting New Performance Feature for Serial ATA*.

Myoungsoo Jung and Mahmut Kandemir. 2012. An evaluation of different page allocation strategies on high-speed SSDs. In *Proceedings of the 4th USENIX Workshop on Hot Topics in Storage and File Systems*.

Myoungsoo Jung and Joonhyuk Yoo. 2009. Scheduling garbage collection opportunistically to reduce worst-case I/O performance in solid state disks. In *Proceedings of IWSSPS*.

Myoungsoo Jung, Ellis Herbert Wilson III, David Donofrio, John Shalf, Mahmut Kandemir. 2012. NANDFlashSim: Intrinsic latency variation aware NAND flash memory system modeling and simulation at microarchitecture level. In *Proceedings of IEEE 28th Symposium on Mass Storage Systems and Technologies (MSST)*.

Chulbum Kim, Jinho Ryu, Taesung Lee, Hyeonggon Kim, Jeawoo Lim, Jaeyong Jeong, Seonghwan Seo, Hongsoo Jeon, Bokeun Kim, Inyoul Lee, Dooseop Lee, Pansuk Kwak, Seongsoon Cho, Yongsik Yim, Changhyun Cho, Woopyo Jeong, Jinman Han, Dooheon Song, Kyehyun Kyung, Youngho Lim, and Younghyun Jun. 2011. A 21nm high performance 64Gb MLC NAND flash memory with 400MB/s asynchronous toggle DDR interface. In *Proceedings of VLSIC*.

Youngjae Kim, Brendan Tauras, Aayush Gupta, and Bhuvan Urgaonkar. 2009. FlashSim: A simulator for NAND flash-based solid-state drives. In *Proceedings of SIMUL*.

LBNL. A new breed of supercomputers for improving global climate predictions. http://crd.lbl.gov/groups-depts/ftg/projects/current-projects/green-flash/.

Jaesoo Lee, Kangho Roh, Wonhee Cho, Hojun Shim, Youngjoon Choi, Jaehoon Heo, Jehyuck Song, Seungduk Cho, Seontaek Kim, Moonwook Oh, Jongtae Park, Wonmoon Cheon, Chanik Park, and Yangsup Lee. 2009. Memory system and method of accessing a semiconductor memory device. In *US20110302352 A1*.

Sungjin Lee, Keonsoo Ha, Kangwon Zhang, Jihong Kim, and Junghwan Kim. 2009. FlexFS: A flexible flash file system for MLC NAND flash memory. In *Proceedings of USENIX ATC*.

Seungjae Lee, Youngtaek Lee, Wookkee Han, Donghwan Kim, Moosung Kim, Seunghyun Moon, Hyunchul Cho, Jungwoo Lee, Daeseok Byeon, Youngho Lim, Hyungsuk Kim, Sunghoi Hur, and Kangdeog Suh. 2004. A 3.3V 4Gb four-level NAND flash memory with 90nm CMOS technology. In *Proceedings of ISSCC*.

Sang-Won Lee, Bongki Moon, Chanik Park, Jae-Myung Kim, and Sang-Woo Kim. 2008. A case for flash memory SSD in enterprise database applications. In *Proceedings of SIGMOD*.

Kaoutar El Maghraoui, Gokul Kandiraju, Joefon Jann, and Pratap Pattnaik. 2010. Modeling and simulating flash based solid-state disks for operating systems. In *Proceedings of WOSP/SIPEW*.

Micron Technology, Inc. 2007. NAND flash memory MLC datasheet, MT29F8G08MAAWC, MT29F16G08Q ASWC. http://www.micron.com/.

Micron Technology, Inc. 2009. NAND flash memory datasheet, MT29F16G08ABABA, MT29F32G08AFABA. http://www.micron.com/.

Vidyabhushan Mohan, Trevor Bunker, Laura Grupp, Sudhanva Gurumurthi, Mircea R. Stan, and Steven Swanson. 2013. Modeling power consumption of NAND flash memories using flashpower. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32, 7 (2013), 1031–1044.

ONFI Working Group. 2011. Open NAND flash interface.

Seonyeong Park, Euiseong Seo, Jiyong Shin, Seungryoul Maeng, and Joonwon Lee. 2010. Exploiting internal parallelism of flash-based SSDs. *Computer Architecture Letters* 9 (2010), 9–12.

David A. Patterson. 2004. Latency lags bandwidth. *Communications of the ACM* 47 (2004), 71–75.

Frankie F. Roohparvar. 2007. Single level cell programming in a multiple level cell non-volatile memory device. U.S. Patent 20070133249 A1.

SNIA. 2006. IOTTA repository. http://iotta.snia.org/.

Michael F. Wehner, Leonid Oliker, John Shalf, David Donofrio, Leroy A. Drummond, Ross Heikes, Shoaib Kamil, Celal Konor, Norman Miller, Hiroaki Miura, Marghoob Mohiyuddin, David Randall, and Woo-Sun Yang. 2011. Hardware/software co-design of global cloud system resolving models. In *JAMES*, Vol. 3.

Michael Wei, Laura M. Grupp, Frederick E. Spada, and Steven Swanson. 2011. Reliably erasing data from flash-based solid state drives. In *Proceedings of USENIX FAST*.

Received March 2014; accepted October 2014