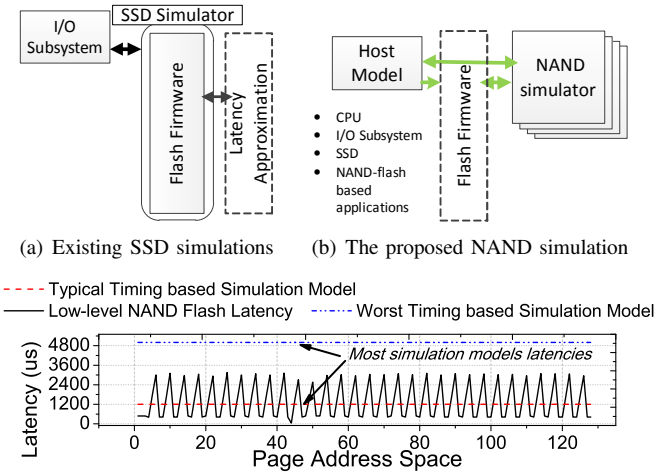
NANDFlashSim: Intrinsic Latency Variation Aware NAND Flash Memory System Modeling and Simulation at Microarchitecture Level

Myoungsoo Jung[1], Ellis Herbert Wilson III[1], David Donofrio[2], John Shalf[2] Mahmut Taylan Kandemir [1]
[1] Department of CSE, The Pennsylvania State University
[2] National Energy Research Scientific Computing Center, Lawrence Berkeley National Laboratory
{mj@cse.psu.edu, ellis@cse.psu.edu, ddonofrio@lbl.gov, jshalf@lbl.gov, kandemir@cse.psu.edu}

***Abstract*—As NAND flash memory becomes popular in diverse areas ranging from embedded systems to high performance computing, exposing and understanding flash memory's performance, energy consumption, and reliability becomes increasingly important. Moreover, with an increasing trend towards multiple-die, multiple-plane architectures and high speed interfaces, high performance NAND flash memory systems are expected to continue to scale. This scaling should further reduce costs and thereby widen proliferation of devices based on the technology. However, when designing NAND flash-based devices, making decisions about the optimal system configuration is non-trivial because NAND flash is sensitive to a large number of parameters, and some parameters exhibit significant latency variations. Such parameters include varying architectures such as multi-die and multi-plane, and a host of factors that affect performance, energy consumption, diverse node technology, and reliability. Unfortunately, there are no public domain tools for high-fidelity, microarchitecture level NAND flash memory simulation in existence to assist with making such decisions. Therefore, we introduce NANDFlashSim; a latency variation-aware, detailed, and highly configurable NAND flash simulation model. NANDFlashSim implements a detailed timing model for operations in sixteen state-of-the-art NAND flash operation mode combinations. In addition, NANDFlashSim models energies and reliability of NAND flash memory based on statistics. From our comprehensive experiments using NANDFlashSim, we found that 1) most read cases were unable to leverage the highly-parallel internal architecture of NAND flash regardless of the NAND flash operation mode, 2) the main source of this performance bottleneck is I/O bus activity, not NAND flash activity itself, 3) multi-level-cell NAND flash provides lower I/O bus resource contention than single-level-cell NAND flash, but the resource contention becomes a serious problem as the number of die increases, and 4) preference to employ many dies rather than to employ many planes promises better performance in disk-friendly real workloads. The simulator can be downloaded from http://www.cse.psu.edu/∼mqj5086/nfs.**

I. Introduction

While processors have enjoyed doubled performance every 18 months, and main memory performance increases roughly 7% in the same time frame, non-volatile storage media has been stuck at a standstill for nearly a decade [23]. Many efforts have been made to remedy such a great gap, and NAND flash has positioned itself at the forefront of such efforts. 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 (SSD), Compact Flash, and Flash Cards. NAND flash density is increasing by two to four times every two years [16], which is in turn decreasing its cost and enabling widespread deployment in arenas as diverse as embedded systems and high-performance computing. In addition, by introducing multiple planes and dies, the NAND flash memory is expected to continue in this trend as it experiences the same ease of scaling multiprocessors currently enjoy. As a result of such proliferation, performance, energy consumption, and reliability of NAND flash memory are becoming increasingly important [26]. 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 often 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-case and worst-case [6], deciding on an optimal NAND flash memory system's configuration is non-trivial. Furthermore, NAND flash memory can have many different parameters depending on what memory system types are employed (e.g., single level cells (SLC), multi level cells



(a) Existing SSD simulations (b) The proposed NAND simulation

(c) Latency comparison

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

978-1-4673-1747-4/12/$31.00 © 2013 IEEE

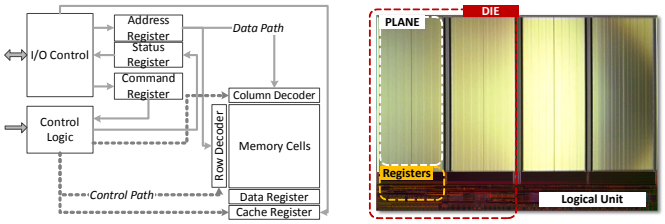
(MLC), diverse node technologies (fabrication processes), page sizes, register management policy, memory density, and pin configurations). Consequently, this large parameter space and sensitivity of NAND flash to such parameters results in memory systems exhibiting significantly different behaviors.

Unfortunately, a comparison between different types of NAND flash memory systems becomes even harder when multi-die and multi-plane architectures are considered [5]. In these architectures, scheduling methods and arbitration [22] among multiple dies and planes are important factors in determining I/O performance [5]. However, incorporation of these methods and arbiters results in a greatly increased complexity of flash firmware and controllers. Even though simulation-based prior research [7], [12], [17], [20] reveals performance tradeoffs in an application level, the main focus of such studies has been on SSD rather than the NAND flash memory system itself; this difference is pictorially shown in Figure 1(a). Such simulations make several assumptions that ignore, to varying extents, the fluctuating timing behaviors of the diverse I/O operations supported by state-of-the-art NAND flash memory. These assumptions range from extremely widespread, where the SSD is modelled as having constant time and energy per I/O request, to more confined but still overly simplified, where dies and planes are modelled but the interactions between various NAND commands and components are still represented with *constants*. This implies that the existing simulation models used in those prior studies are strongly coupled to particular flash firmware and policies – performing the exact same study using slightly different firmware or policy-set has the potential to result in wildly different performances and conclusions. 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 such prior studies are oblivious of *intrinsic latency variation* of NAND flash, they are not be able to properly model diverse node technologies. Also, simplified latency models ignore the substantial contributions of the flash firmware to 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, flash controllers, and so on.

To address heretofore mentioned 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 enable research on the NAND flash memory system itself as well as many NAND flash-based devices, as illustrated in Figure 1(b). Specifically, in this paper, we propose *NANDFlashSim*; a latency variation-aware, detailed and highly reconfigurable μarch-level NAND flash memory system based on multi-die and multi-plane architectures. To the best of our knowledge, NANDFlashSim is the first simulation model to target the NAND flash memory system itself at μarch-level, and the first model to provide sixteen latency variation-aware NAND flash operations with NAND command set architecture.

From our comprehensive experiments using NANDFlashSim, we found that 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 throughputs improvements between quad dies and octal dies, between four-plane and eight-plane, and between 4KB and 8KB page size are 10.9%, 10.8%, and 10.9%, respectively, while the write throughputs are improved by 91.2% on average. 2) the main contributor of performance bottleneck is I/O bus activity, not NAND flash activity itself. 50.5% cycles of total I/O execution cycles are consumed by operations related to such I/O bus activity. The bottleneck is more problematic when *advance NAND flash commands* (e.g., cache and multi-plane mode) are applied. 3) MLC NAND flash provides lower I/O bus resource contention than SLC NAND flash, but such resource contention becomes a serious problem as the number of dies increases, and 4) preference of employing many dies rather than employing many planes provides average 54.5% better performance in terms of throughput in disk-friendly workloads [25]. This paper 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 multi-stage 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 mode, cache mode, internal data move mode, multi-plane mode, multi-plane cache mode, interleaved-die mode, interleaved-die cache mode, interleaved-die multi-plane mode, and interleaved-die multi-plane cache mode. These detailed NAND operation modes and their associated timings expose performance optimization points to NAND flash-based application designers and developers.
- *Intrinsic Latency Variation-Aware Simulation Model:* NAND flash memory, especially MLC, suffers from intrinsic performance variations when accessing a block. In our observations, write latency of [19] and [8] varies between 250μs to 2,200μs and 440μs to 5,000μs, respectively (maximum latencies are 8.8 $\sim$ 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 prototype a NAND flash hardware platform, called 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 Micoarchitecture:* NANDFlashSim supports highly reconfigurable architectures in terms of multiple dies and planes. This architecture allows a researcher to explore true internal parallelism in such an architecture by exposing the intrinsic latency variations in NAND flash. In



(a) NAND flash microarchitecture (b) Die microphotograph [11]

Fig. 2. NAND flash memory system internals.

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.

II. NAND Flash Microarchitecture

Figure 2(a) illustrates the NAND flash microarchitecture [19], and Figure 2(b) depicts the physical NAND memory cell microphotograph [11]. 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 to reduce pin counts, interface complexity, and energy consumption [19]. Because of this, a host model must first inform the NAND flash package that it wishes to use the I/O bus through control logic before acquiring it. This information is provided via control signals like *command latch enable* (CLE) and *address latch enable* (ALE). Similarly, NAND commands are responsible for signalling usage of the I/O bus in addition to classifying 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 adopted to provide collection and buffering for delayed write-back of small writes and to fill the performance gap between the NAND flash interface and flash memory cells. Supporting multiple registers is a common trend 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 enables the memory system to serve multiple I/O requests simultaneously [14].
- *Die.* A die contains an even number of planes and constitutes a NAND flash package. Depending on how many dies 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 sharing 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 [11] and eight dies, our proposed simulation model can be configured to simulate a much larger number of planes and dies in a logical unit.

III. 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, erase operation executes on an entire block. To operate NAND flash memory, the first task is to load a command into the command register by raising the CLE signal, which informs what operation will be executed. After that, a start address for the operations is loaded into an internal address register by raising the ALE signal. Once the address is 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 *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 [5]. Below, we explain different I/O modes, which are used in concert with these legacy commands.

A. Cache Mode Operation

In *cache mode* operation, data are first transferred to a cache register and then copied to a data register. After that the NAND flash memory system enters the TIN stage. In the meantime, the memory system is available again 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 register and that of writing data into the NAND memory cells, thereby hiding the TIR time. Just like writes, read operations can also take advantage of the cache register. However, in our observations, cache mode operations demonstrate slightly different performances between reads and writes. This is due to the latency-dominating NAND operation differing between, which will be further discussed in Section VIII.

B. Internal Data Move Mode Operation

Flash 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 suprisingly expensive and time consuming. To reduce the time required to copy data, state-of-the-art

NAND flash memory support *internal data move* operations. In these operations, a host is not involved in managing the data copy process. Instead, the host only has 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 [4], a critical task of flash firmware.

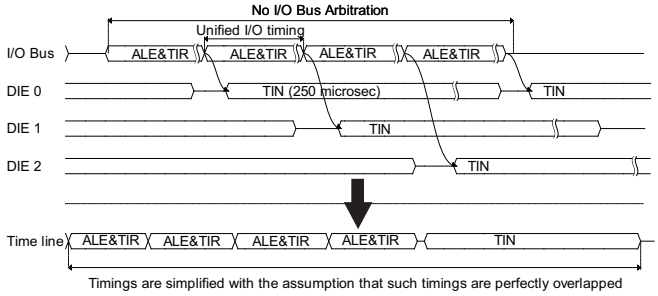
C. Multi-plane Mode Operation

Multi-plane mode operations serve I/O requests using several planes at a time 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 multi-plane architecture carries with it limitations for addressing planes. Specifically, in multi-plane mode operations, I/O requests should indicate the same page offset in a block, same die address, and should have different plane addresses [19], [21]. These constraints are collectively referred to as the *plane addressing rule*. Therefore, performance enhancement using a multi-plane architecture may be limited based on user access patterns (we will discuss this issue in Section VIII-E). Regulating plane addressing rules is required to obtain realistic performance with the multi-plane mode of operation. Using such rules, NANDFlashSim provides an accurate implementation of multi-plane mode operations, which may be used in any combination with other NAND flash operations.

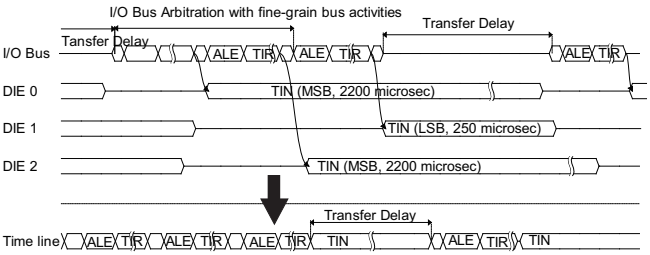
D. Interleaved Die Mode Operation

State-of-the-art flash memory share 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, I/O bandwidth of the system also reduces. This is because all NAND operations except those related to NAND memory cells (e.g., TON, TIN) should acquire 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 way to share the I/O bus and take advantage of internal parallelism by interleaving NAND operations among multiple dies. Unlike multi-plane mode operations, interleaved-die mode operations have no addressing restrictions.

It should be noted that all NAND operations discussed above can be used in any combination with interleaved-die operations. For example, a host model can issue an interleaved-die multi-plane mode operation, which stripes a set of multi-plane 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 being



(a) A typical-case timing parameter based simulation model



(b) Latency-aware NANDFlashSim

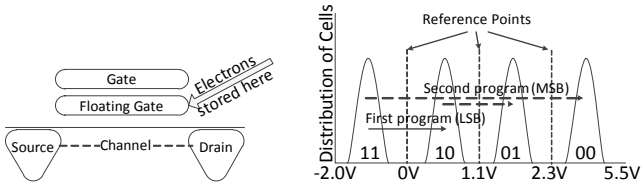
Fig. 3. A timing diagram of interleaved die with four legacy writes. While an existing simulation model simplifies bus activities and assumes that latencies are perfectly overlapped and interleaved with constant time, NANDFlashSim employs fine-grain bus activities and is aware of intrinsic latency variations.

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 operations. Furthermore, intrinsic latency variations exhibited by the NAND flash make it difficult for a latency model with constant time 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 will calculate the latency under the assumption that *timings are perfectly overlapped and interleaved*. Let t_{io} denote execution time for I/O activities, and t_{prog} denote programing (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 time line 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.

IV. Intrinsic Latency Variation of NAND Flash

NAND flash memory has the interesting characteristic of performance variation [6], [13], [14], which results in the latencies of the NAND flash memory system to fluctuate significantly depending on the address of the pages in a block. Typically, this variation is not specified in the datasheets of



(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 [14].

NAND flash memory. NAND flash memory puts electrons, which represents cell states, into a NAND flash floating gate. To achieve this, NAND flash memory selects the NAND flash memory cells, and makes an electron channel between a source and drain (see Figure 4(a)). Once the channel is built and voltage is applied over a certain threshold voltage, electrons can be moved from the channel to the floating gate. This process is called *Fowler-Nordheim tunneling* (*FN-tunneling*) [14], which is a well-known programming (write) operation. As illustrated in Figure 4(b), based on differing cell distributions, a MLC NAND flash memory system can identify bit states like '11', '10', '00' and '01' in a cell[1]. According to the specific bit states for programming, therefore, a MLC NAND flash memory system will end up spending 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)* [13], [14].

For example, in the first phase, MLC NAND flash programs a cell from '11' to '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 for MSB pages requires more energy and takes a longer time when compared to LSB pages [6], [15], [24]. Due to these NAND flash memory characteristics, one may observe performance variations between worst-case and typical-case programming time parameters. Since there is no need for ISPP to a specific cell in SLC flash, this latency variation characteristic is more pronounced in MLC NAND flash memory.

V. Related Work

There are several prior studies for simulating a NAND flash-based SSD. The SSD add-on [20] to DiskSim [3] is a popular simulator that models idealized flash firmware. FlashSim [12] is another simulator, implemented using object-oriented code for programmatic ease and extensibility. This simulator supports several types of flash software algorithms. While these simulation models compute performance by calculating latency for each of the basic NAND operations, SSDSim [7] accommodates latency calculation models for cache, multi-plane, interleaved-die operations of SLC devices at application-level.

Even though these simulation models can enable researchers to explore the design tradeoffs of SSDs, they have limitations

[1]The '0' bit in a NAND flash cell represents programed (written) state.

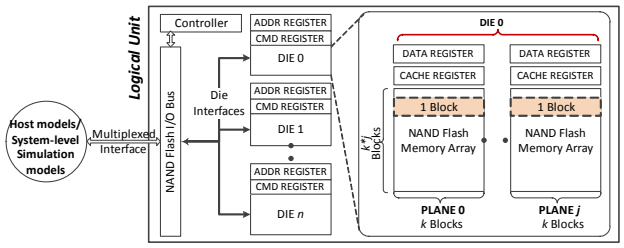


Fig. 5. NANDFlashSim architecture. NANDFlashSim is a reconfigurable μ-level multi-plane and multi-die architecture. The number of registers, blocks, planes and dies can be reorganized, and each entity has independent synchronized clock domain.

in simulating the μarch-level NAND flash memory since they highly simplify NAND flash characteristics, latencies, and energies from a flash firmware perspective. Also, *these studies are appropriate to simulate only SLC NAND flash type*.

•**Unaware of latency variations.** These existing simulation models are ignorant of NAND flash memory's latency variations; they implement the flash memory system based on constant times and energies of worst-case or typical-case time parameters. However, as mentioned in Section III, the state-of-the-art memory systems are very complex and support diverse NAND I/O operations for high performance I/O, which results in latency varying immensely even between executions of operations of the same type. In contrast, our proposed NANDFlashSim is aware of the latency variations based on most significant bit (MSB) and least significant bit (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 μarch-level. As consequence, NANDFlashSim is able to simulate both *MLC and SLC NAND flash*.

•**Coarse-grain NAND command handling.** Moreover, these past studies mimic multi-die and multi-plane architecture using coarse-grain I/O operations, which means that NAND operation and control are simplified by host-level I/O requests. Even though they consider basic I/O timing based on time parameter statistics and internal parallelism of NAND flash memory, the evaluation of accurate memory system latencies is non-trivial. Using multi-stage and command chains for each of the NAND flash operations , our proposed NANDFlashSim, reconfigurable for multi-die and plane architectures, provides detailed timing models for NAND operations and manages bus arbitration based on different latencies at μarch-level.

•**Weak model of NAND flash memory constraints.** The memory system's performance and energy consumption can exhibit a variety of patterns due to NAND flash memory constraints. For example, as mentioned in Section III, multi-plane I/O operations should satisfy plane addressing rules. This constraint results in different performance characteristics depending on I/O patterns. Even though the past studies consider these kinds of constraints, their simulation is *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 in μarch-level, and is not tied to any specific flash firmware, algorithm or NAND flash applications like SSDs.

VI. 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 algorithm 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/develop flash software for their specific purposes. To achieve its design goals, instead of employing underlying simplified latency calculation models, NANDFlashSim uses a NAND command set architecture and individual state machines associated to the command sets, which results in independent synchronous clock domains. These mechanisms enable designers and architects to closely study the NAND flash performance and optimization points at a cycle-level by exposing the details of NAND flash.

A. Software Architecture

Figure 5 illustrates the software architecture of our proposed simulation model. NANDFlashSim is comprised of a logical unit, 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 operations 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 Open NAND Flash Interface (ONFI) [21]. 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 make the NAND memory system malfunction, NANDFlashSim verifies the correctness of command uses by checking the command/address registers and its own state machines every cycle. If it detects a wrong 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 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 how 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. Once 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. Note that this is handled by NANDFlashSim instead of the host model. In the meantime, 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* (we will discuss this shortly). It should be noted that using these two interfaces, other simulator models can be easily integrated into NANDFlashSim.

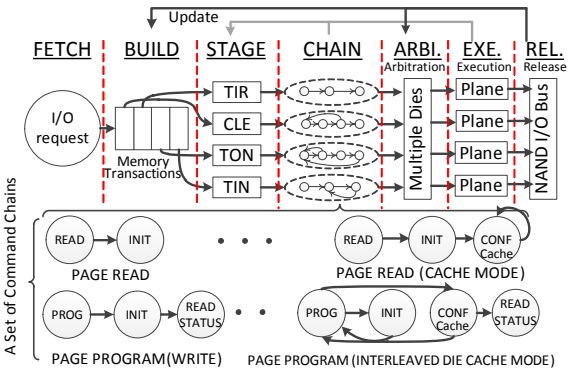


Fig. 6. The process of NAND flash memory transactions and examples of NAND command chains. NANDFlashSim handles fine-grain NAND transactions by NAND command set architecture

B. Clock Domains and Lifetime of Transaction

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) NAND flash memory system's clock domain. The entities of NANDFlashSim are updated at every clock cycle, and the transaction lives until either getting 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 cycles 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 not worth simulating. In this mechanism, NANDFlashSim looks over all modules in the logical unit, and then finds out 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 the meaningless cycles in the update process.

VII. Implementation Details

A. NAND Command Set Architecture

The number of combinations of operations possible with a state-of-the-art NAND flash memory is as high as sixteen, 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 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 *multi-stage operations* and *command chains*, as described next.

Multi-stage Operations. Stages are defined by 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 by CLE, ALE, TIR, TOR,

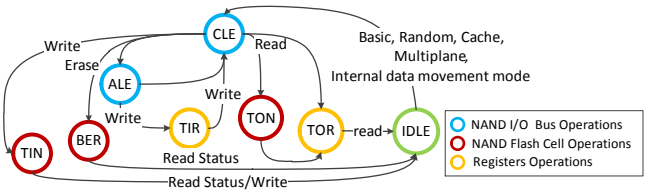


Fig. 7. State machine for multiple NAND stages. Each state is identified by different type of stages and states of the machine are transited by different type of NAND commands. Since each die has their own state machine, NANDFlashSim provides an independent clock cycle domain per die.

TIN, TON, and BER. CLE is a stage for a command by following command latch enable signal, and ALE 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 the name described earlier in Section II. These stages comprise an independently clocked synchronous state machine, as illustrated in Figure 7. This state machine describes different stages for each NAND I/O operations as visualized in the bottom of Figure 6. All dies have such state machines based on stage and regulate/validate correctness of NAND commands and multi-stage sequence.

Command Chains. A command chain is a series of NAND commands, and each combination of NAND operations has its own command chain. Even though the state machine with multi-stage 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. Also, the process of transitioning from one stage to another stage varies based on what command is loaded into the command register. For example, as illustrated in Figure 6, the write operation has a different sequence for data movement and a different number of commands compared to the erase and read operations. When a combination of NAND operations with cache, multi-plane or die-interleaved mode is applied, the differences are more striking. Therefore, NANDFlashSim employs command chains, which are updated by the NANDFlashSim controller and logical unit. Also, 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. Using multi-stage operations and command chains, NANDFlashSim defines a NAND command set architecture and provides a cycle accurate NAND flash model.

B. Awareness of Latency Variation

NANDFlashSim is designed to be aware of intrinsic latency variations when it simulates MLC NAND flash. To extract real performance 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. Suppose that n_{pages} is the number of page per block, and λ is constant value related to a page offset. n_{pages} and λ are device specific value. Typically, n_{pages} is powers of two, and λ is 2 or 4. We assume that a set of page addresses, which show relatively high latency, indicates the MSB pages referred as to $msb(n)$, where $\forall n, 0 \leq n \leq (n_{pages}/2)$. We also assume that another set of page addresses, which exhibit low latencies, are the LSB pages referred as to $lsb(n)$. With this assumption[2] in place, NANDFlashSim generates different programming timing based on these two sets of page addresses, which are extracted from MSIS. Even though these address sets of page addresses can be varied based on NAND flash manufacturers, we found that these address sets can be classified by two groups for diverse NAND flash devices (we tested eight devices from four manufacturers, and technology nodes of them range from 24 nanometer to 32 nanometer). These two groups of such page address sets indicated by different subscripts, α and β (e.g., $msb_\alpha(n)$, $lsb_\alpha(n)$, $msb_\beta(n)$, and $lsb_\beta(n)$). For $lsb_\alpha(n)$, if n is zero or equal to $n_{pages}-1$ then n and $n+1$ are LSB pages. Otherwise, the $lsb_\alpha(n)$ is generated by λn, and the $msb_\alpha(n)$ is generated by $\lambda n-(\lambda+1)$. On the other hand, for $lsb_\beta(n)$, if n is less than λ or n is grater than $n_{pages}-\lambda$ then n is LSB pages. Otherwise, λn and $\lambda n+1$ are elements of $lsb_\beta(n)$, and $\lambda n-(\lambda+2)$ and $\lambda n-(\lambda+1)$ are elements of $msb_\beta(n)$. It should be noted that NAND flash parameters related to these sets of addresses only affect NAND flash activity, especially transfer-in of NAND (TIN) stage. I/O bus activities such as CLE, ALE, TIR, and TOR are not affected by such sets of addresses.

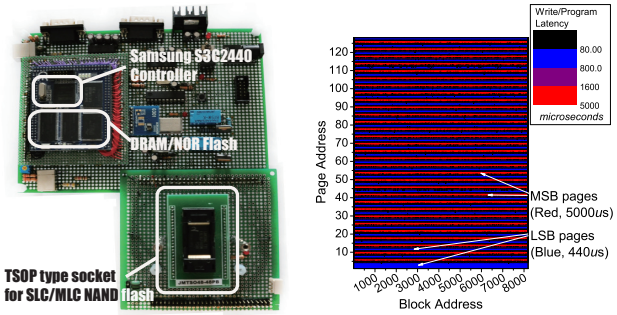
C. Enforcing Reliability Parameters

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 NANDFlashSim records the number of programs for each page. If a request tries to program a page over the NOP limit, NANDFlashSim informs the host model of this violation. In addition, the plane model maintains the page address which was most recently written. When a host model requests to program a page that has a lower address than the most recently written page address, NANDFlashSim reports this as a violation of the in-order update constraint to the host. To enforce the endurance limitation, each block in the plane model tracks erase counts. When NANDFlashSim detects a request that would erase a block over the number of erases that the memory system guarantees, it informs the host model of this endurance violation. These reliability models provide an environment for system designers and architects to study NAND flash reliability and explore future research directions such as developing new wear-leveling, garbage collection and address mapping algorithms.

VIII. Evaluation

For the validation of NANDFlashSim compared to other real products, we utilize two different types of MLC NAND flash packages [19] (i.e., Single Die Package (SDP) and Dual Die Package (DDP), and two MLC devices came from two different manufacturers [19], [8]. In addition, for evaluating

[2]This assumption is already widely used by both industry and academia [6], [15], [24].



(a) Our Hardware Prototype (MSIS) (b) A 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.

Device Type	Feature	Value
Single Level Cell	Page Size(Byte)	2048
	# of Page Per Block	64
	# of Block	4096
	Write Latency(us)	250
	Read Latency(us)	25
	Erase Latency(us)	1500
Multi Leve Cell 1 (MLC1)	Page Size(Byte)	2048
	# of Page Per Block	128
	# of Block	8196
	Write Latency(us)	250∼2200
	Read Latency(us)	50
	Erase Latency(us)	2500
Multi Leve Cell 2 (MLC2)	Page Size(Byte)	8192
	# of Page Per Block	256
	# of Block	8196
	Write Latency(us)	440∼5000
	Read Latency(us)	200
	Erase Latency(us)	2500

TABLE I
NAND FLASH DEVICE CHARACTERIZATION

NANDFlashSim, we also use SLC and MLC type NAND flash. The main parameters for those devices such as block, page sizes and latency, are described in Table I. Unless otherwise stated, we will use parameters of MLC1 as default.

Table II analyzes workloads that we tested. In addition to a number of disk-friendly traces from actual enterprise applications (msnfs, fin, web, usr, and prn) [25], we also synthesized write and read intensive workloads of which access pattern are fully optimized to NAND flash. Specifically, in the swr and srd workloads, we perform reads and writes of data on different block boundaries, and make 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 less restrictions. Access patterns of all workloads tested are used by both the hardware prototype (MSIS) and NANDFlashSim.

A. Validation of NANDFlashSim

Latency Validation. Figure 9 pictorially illustrates cumulative distribution function (CDF) of latency for both NANDFlashSim and MSIS on enterprise application workloads. In this validation, interleaved die mode and multiplane mode NAND commands are interplayed with legacy mode NAND operations, and a queue (32 entries) [9] is applied for handling incoming I/O requests. The microscopic illustration of inflections for each CDF are also pictorially embedded. In the figures, the red line represents MSIS latency with MLC2 [8] and the blue line represents latency of variation-aware NANDFlashSim. As shown in the figures, latencies of NANDFlashSim are almost completely overlapped with the real product latencies. Since NANDFlashSim employs a variation-aware timing model as default, it exhibits very close performance to the real product of MSIS.

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

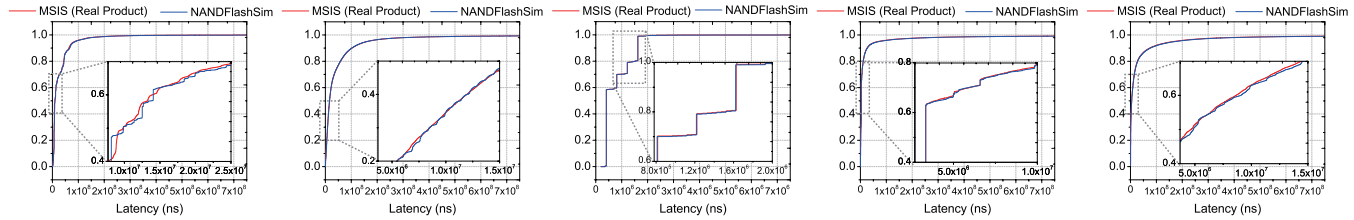
TABLE II
WORKLOADS CHARACTERIZATION

System Performance Validation. In these performance validation tests, we evaluate performance for our variation-aware based NANDFlashSim, worst-case timing based simulation model [10], typical-case timing based simulation model [20], [12], [7] 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 striping them across multiple dies.

Figure 10 compares the SDP [19] read/write performance on NANDFlashSim and MSIS. The throughput values obtained using NANDFlashSim (with variance-aware latency model) are close to the real product performance of MSIS. In the read cases, NANDFlashSim is no more accurate relative to MSIS than the other timing models. This is because variation for reads of NAND flash memory is negligible. In contrast, write operations show different performances according to the type of latency models employed. Since write performances are seriously varied between the minimum to maximum latency, there is a performance gap between performance of MSIS and that of both worst-case timing parameter and typical-timing parameter based latency models.

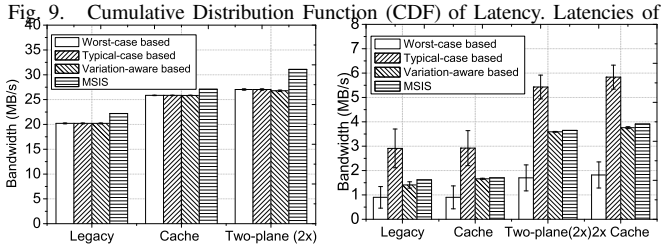
These plots also depict bars of the percentage of deviation between MSIS performances and NANDFlashSim performances among different latency models. Specifically, variation-aware NANDFlashSim provides around 12.9%, 2.1%, 1.6% and 3.8% deviation in performance for the legacy, cache mode, 2x mode, and 2x cache mode operation, respectively. This is a significant improvement from 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 read/write performance results with DDP [19] on the same test set. Typical-timing parameter based latency model shows highly errant performance compared to MSIS ones (deviation range is 83.1% to 170.9%). Even though this latency model is the most popular one among SSD simulators, it mimics the ideal performance, which can be achieved *if and only if* the time spent in TIN can perfectly



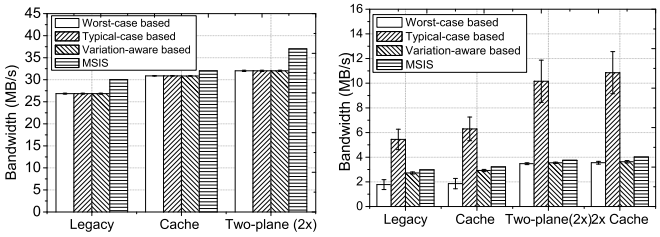
(a) msnfs (b) fin (c) web (d) usr (e) prn

Fig. 9. Cumulative Distribution Function (CDF) of Latency. Latencies of NANDFlashSim are mostly overlapped with real NAND flash product latencies.



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

Fig. 10. Performance comparison on SDP. Typical-case/worst-case time parameter based latency models show unreasonable performance gap from real product ones.



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

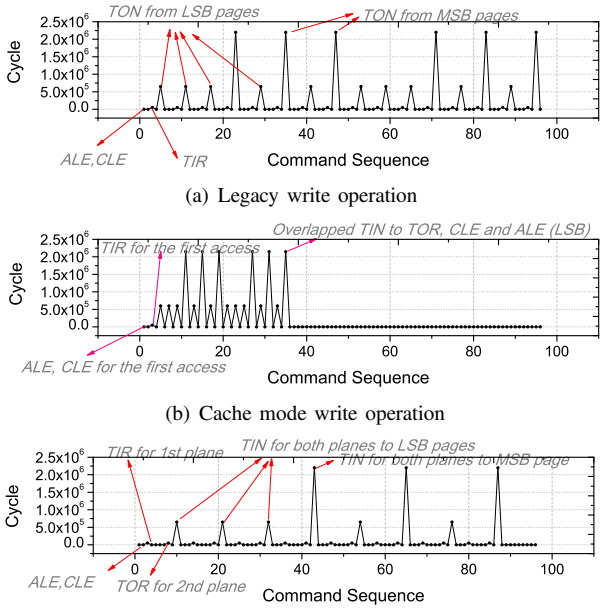
Fig. 11. Performance comparison on DDP. While typical-based latency model show more discrepant performance than that of multi-plane tests, the variation-aware NANDFlashSim provides performances close to the real product ones.

be overlapped with other operations stages, and the NAND bus I/O utilization is reasonably high across multiple dies. Although the worst-case parameter based latency model provide closer performance (deviation range is 7.3% $\sim$ 42.0%), it still shows unrealistic performance. 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.

B. Individual Cycle Analysis

Figures 12 and 13 illustrate an individual cycle comparison among legacy, cache mode, and 2x mode operations. In this evaluation, we request 8 pages read or write for two blocks, and the size of the requests is 2KB.

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

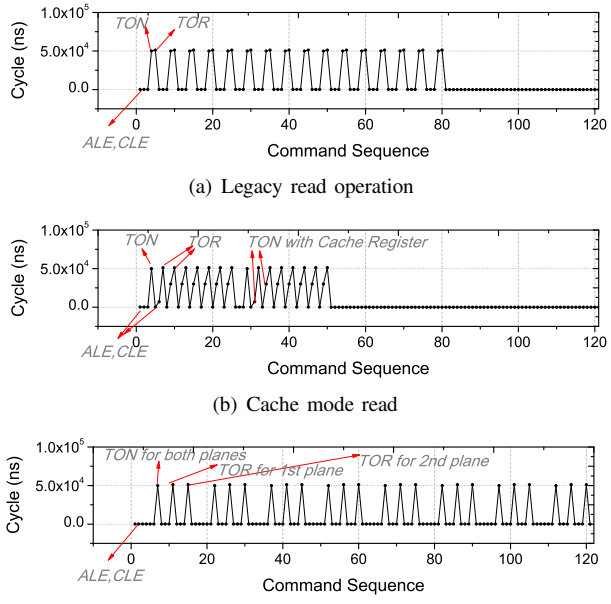


(a) Legacy write operation

(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 a chronological oder based on a set of NAND commands that host commits, and one cycle takes 1 nanosecond.



(a) Legacy read operation

(b) Cache mode read

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

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

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

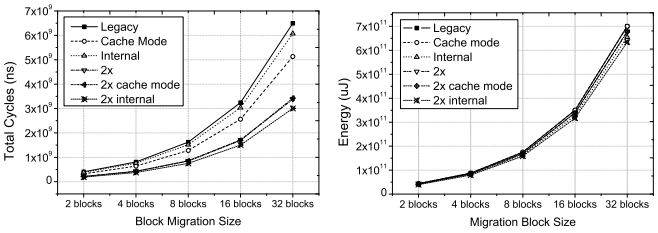
Read Cycles. Figure 13 illustrates read cycle analysis executed by NANDFlashSim. Read operation behaviors for legacy, cache mode and 2x modes are similar to the writes, having only two main differences: 1) latencies for the TON operation do not fluctuate like TIN of write operations, and 2) the TOR cycle fraction of the total execution cycles (see Figure 13(a)) is close to the TON ones. In Figure 13(b), one can see that cycles for TOR, which are related to bus operations, are higher than TON related to operations for NAND memory cells, meaning that reads spend many cycles on the I/O bus operations (we will discuss more detail in Section VIII-D).

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

C. Performance and Power Consumption Comparison: Page Migration Test

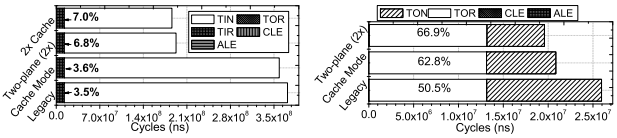
We also evaluate a page migration test, which is a series of tasks copying pages from source block(s) to destination block(s) and erasing the block(s). This test mimics a very time consuming job of a flash firmware occuring frequently during garbage collection. To evaluate performance of page migration, 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 are done. These page migration tasks are performed by legacy, cache, internal, 2x, 2x cache, and 2x internal mode operations. As shown in Figure 14(a), there is little performance difference at page migrations of 2 blocks, but as the number of blocks for the migration increases, latencies for 2x cache mode, internal data move, and 2x internal data move mode operations are about two times smaller than legacy, cache mode and 2x mode operations. Importantly, the 2x internal data move mode operation outperforms all other operations. In contrast, energy consumptions for each NAND I/O operation are not much different between them as shown in Figure 14(b). This is because even though the latency benefits come from internal parallelism, the same amounts of power for operating I/O requests are required by all memory system components.

Figure 16 shows 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.



(a) Bandwidth (b) Energy

Fig. 14. Block migration performance comparison and energy consumption. While energy consumption are similar to different NAND operation modes, 2x cache mode and internal data move mode operations have great impact on enhancing performance in the page migration test.



(a) Writes (swr) (b) Reads (srd)

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

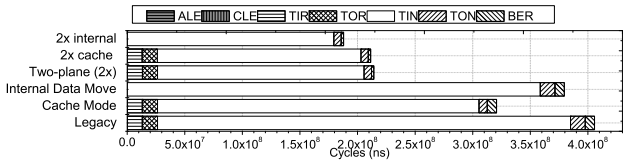


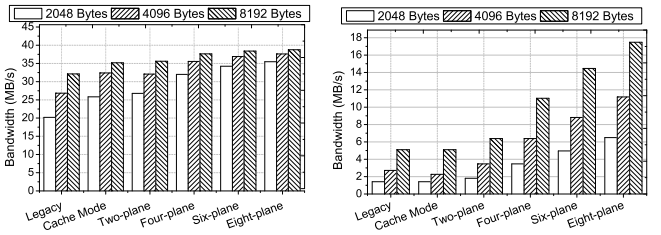
Fig. 16. Cycle analysis for page migration. Internal data move modes removes the most NAND I/O bus activities thereby improving throughputs.

D. 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 spend at most 7.0% of the total time performing data movement (TOR/TIN), read operations spend at least 50.5% of the total time doing so. Therefore, even though 2x or cache mode is applied to read operations (see Figure 15(b), there is small benefit in terms of bandwidth. We believe that this is a reason why much research in industry is directed towards enhancing bus bandwidth. However, do note that the write performance cannot be enhanced by any kind of high speed interfaces because the speed is dominated by the latency of the TIN stage. It also should be noted that since NANDFlashSim allows us to count cycles dedicated to each NAND flash stage and command, it helps us determine which operations are the performance bottleneck, or which operation is the best operation for a specific access pattern to improve performance.

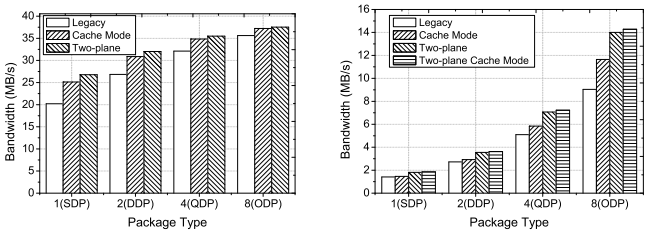
E. Performance on Multi-plane and Multi-die Architectures

Multi-plane. Figure 17 compares throughputs observed in NANDFlashSim for varying the numbers of planes and transfer sizes. Performance of write operations is significantly enhanced as the number of planes increases because most of TIN can be executed in parallel. In contrast, for the read operations, such benefits are much lower than for write operations. This is because cycles for data movement (TOR) are a dominant factor in determining bandwidth, and it is unaffected



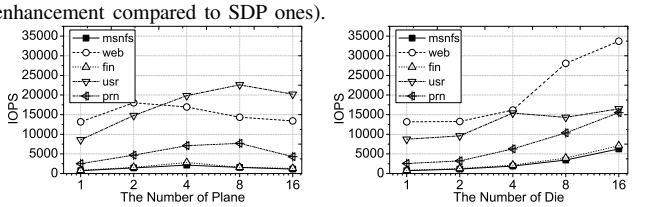
(a) Read performance (srd) (b) Write performance (swr)

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



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

Fig. 18. Multi-die architecture performance. While many dies architecture with the swr workload enjoys linear enhancement (ODP improves throughput 541.1% to the SDP ones), it saturates read throughputs with eight dies (76.3% enhancement compared to SDP ones).



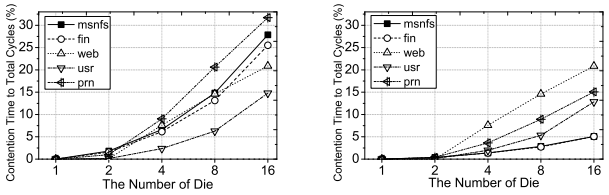
(a) Sensitivity to number of plane (b) Sensitivity to number of die

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

by the number of planes. 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 hard to build multi-plane mode operations with existing disk-friendly I/O access patterns.

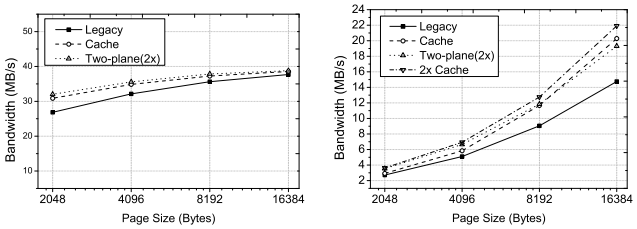
Multi-die. Figure 18 illustrates performance improvement as the number of dies increase. In this section, we tied multiple dies to one NAND flash I/O bus path and have them sharing one CE pin. Similar to multi-plane operations, reads performance enhancement (as the number of dies increases) is limited by latency of the TOR operation. Even though multiple dies are able to serve I/O requests in parallel, performance is bounded by data movement again. This is because during execution of a TOR operation, the NAND flash I/O bus is not capable of handing another TOR one. Therefore, regardless of the fact that 2x or cache mode are applied, TOR operations are the performance bottleneck in read case.

In contrast, as shown in Figure 18(b), throughputs of write



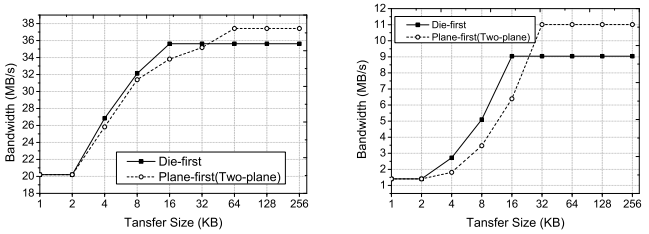
(a) Single level cell (SLC) (b) Multi level cell (MLC)

Fig. 20. Resource contention comparison between SLC NAND and MLC NAND devices. The resource contentions of MLC NAND flash have less impact on SLC NAND flash, but the contention problem is still problematic and become more serious as the number of die increases.



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

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



(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, performance enhancement with multi-die and plane architecture show different performance.

operations are significantly improved by increasing the number of dies. The reason behind this benefit is interleaving 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 multi-stage operations and being aware of intrinsic latency variations at 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.

F. Performance Sensitivity to Page Size

Intuitively, large page sizes can be a good choice to achieve high bandwidth because many bytes can be programmed or read within the same amount of cycles. However, this intuition is only true for writes. Figure 21 plots performance sensitivity to different page sizes on 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 VIII-E, the small enhancements for read operations are due to bus resource conflicts and the large time spent in data movement.

G. 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 execute simultaneously. Instead, this I/O bus activities can be interleaved across multiple dies at 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 to 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 MLC's ones. The reason is that since the latencies of MLC activities are much longer than the latencies of SLC activities, it has more chances to be executed with I/O bus activities at the same time. However, as the number of die increases, for both SLC and MLC throughput, the fraction of the I/O bus resource contention to total I/O execution time increases, which is a reason of performance limitation in many dies architecture.

H. Scheduling Strategy

To test the potential research on NAND command scheduling strategies, we implemented two simple command schedulers in the logical unit of NANDFlashSim: 1) Die-first and 2) Plane-first schedulers. 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. Even though plane-first operation provides better performance, the die-first scheduler is more flexible in serving I/O requests of a smaller size. This is because multi-plane operation performance is limited by plane addressing rules (see Section III-C), whereas multiple dies can be interleaved to serve I/O requests without any addressing constraints.

IX. 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 virtualized dual core, 1GB memory, and 200GB disk. The source code can be downloaded from http://www.cse.psu.edu/∼mqj5086/nfs.

X. Conclusion

Since NAND flash memory is sensitive to a large number of parameters, and some performance parameters have significant latency variation, making decisions on how to configure NAND flash memory for optimal performance is non-trivial. A comparison of various NAND flash memory architectures become even harder when considering multi-die and multi-plane architectures, latency variations, energy consumption costs, reliability issues, and addressing restrictions. Therefore in this work we propose NANDFlashSim, a detailed and highly configurable low-level NAND flash simulation model. NANDFlashSim supports detailed timing models for sixteen I/O operations by being aware of intrinsic latency variations. Our ongoing work includes incorporating a 400MHz high speed NAND interface (not published yet) and implementing a multiple logical unit on chip architecture. In addition, we plan to apply our simulation model to cycle accurate Green Flash [1] and Tensilica Xtensa simulation model [2] of hardware/software co-design platform for exascale computing [18].

XI. Acknowledgements

We would like to express thanks to Dean Klein (Micron Technology, Inc.), Seung-hwan Song (University of Minnesota), and Yulwon Cho (Stanford University) for technical support/discussion on NAND flash memory technologies. We are grateful to many anonymous reviewers for their detailed comments which have greatly improved the quality of our paper. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

References

[1] http://www.lbl.gov/cs/html/greenflash.html.
[2] http://www.tensilica.com/products/hw-sw-dev-tools/.
[3] Bucy, J. S., et al. The disksim simulation environment version 4.0 reference manual. In *Parallel Data Laboratory* (2008).
[4] Chang, L.-P., and Kuo, T.-W. Real-time garbage collection for flash-memory storage systems of real-time embedded systems. *ACM Transactions on Embedded Computing Systems 3*, 4 (November 2004).
[5] Fisher, R. Optimizing NAND flash performance. In *FlashMemory Summit* (2008).
[6] Grupp, L. M., et al. Characterizing flash memory: Anomalies, observations,and applications. In *MICRO* (2009).
[7] Hu, Y., et al. Performance impact and interplay of SSD parallelism through advanced commands,allocation strategy and data granularity. In *ICS* (2011).
[8] Hynix, Inc. NAND flash memory MLC datasheet, H27UBG8T2A. In *http://www.hynix.com/* (2009).
[9] Intel, and Seagate. *Serial ATA Native Command Queuing: An Exciting New Performance Feature for Serial ATA*. 2003.
[10] Jung, M., and Yoo, J. Scheduling garbage collection opportunistically to reduce worst-case I/O performance in solid state disks. In *Proceedings of IWSSPS* (2009).
[11] Kim, C., et al. A 21nm high performance 64gb mlc nand flash memory with 400mb/s asynchronous toggle ddr interface. In *VLSIC* (2011).
[12] Kim, Y., et al. Flashsim: A simulator for NAND flash-based solid-state drives. In *SIMUL* (2010).
[13] Lee, J., et al. Memory system and method of accessing a semiconductor memory device. In *US2009/0310408A1* (2009).
[14] Lee, S., et al. A 3.3v 4gb four-level NAND flash memory with 90nm cmos technology. In *ISSCC* (2004).
[15] Lee, S., et al. Flexfs: A flexible flash file system for MLC NAND flash memory. In *FAST ATC* (2009).
[16] Lee, S.-W., et al. A case for flash memory SSD in enterprise database applications. In *SIGMOD* (2008).
[17] Maghraoui, K. E., et al. Modeling and simulating flash based solid-state disks for operating systems. In *WOSP/SIPEW* (2010).
[18] Michael F. Wehner and others. Hardware/software co-design of global cloud system resolving models. *JAMES* (2011).
[19] Micron Technology, Inc. NAND flash memory MLC datasheet, MT29F8G08MAAWC, MT29F16G08QASWC.
[20] N. Agrawal et al. Design tradeoffs for SSD performance. In *Proceedings of USENIX ATC* (2008).
[21] ONFI Working Group. Open NAND flash interface.
[22] Park, S.-Y., et al. Exploiting internal parallelism of flash-based SSDs. In *Computer Architecture Letters* (2010).
[23] Patterson, D. A. Latency lags bandwidth. In *CACM* (2004).
[24] Roohparvar, F. F. Single level cell programming in a multiple level cell non-volatile memory device. In *U.S. Patent 7529129* (2007).
[25] SNIA. http://iotta.snia.org/. IOTTA repository.
[26] Wei, M., et al. Reliably erasing data from flash-based solid state drives. In *FAST 11* (2011).