Understanding the Impact of Emerging Non-Volatile Memories on High-Performance, IO-Intensive Computing

Adrian M. Caulfield* Joel Coburn* Todor I. Mollov* Arup De* Ameen Akel*
Jiahua He*† Arun Jagatheesan†
Rajesh K. Gupta* Allan Snavely† Steven Swanson*
*Department of Computer Science and Engineering, University of California, San Diego
†San Diego Supercomputer Center, University of California, San Diego

***Abstract*—Emerging storage technologies such as flash memories, phase-change memories, and spin-transfer torque memories are poised to close the enormous performance gap between disk-based storage and main memory. We evaluate several approaches to integrating these memories into computer systems by measuring their impact on IO-intensive, database, and memory-intensive applications. We explore several options for connecting solid-state storage to the host system and find that the memories deliver large gains in sequential and random access performance, but that different system organizations lead to different performance trade-offs. The memories provide substantial application-level gains as well, but overheads in the OS, file system, and application can limit performance. As a result, fully exploiting these memories' potential will require substantial changes to application and system software. Finally, paging to fast non-volatile memories is a viable option for some applications, providing an alternative to expensive, power-hungry DRAM for supporting scientific applications with large memory footprints.**

***Index Terms*—storage systems, non-volatile memory, IO performance, flash memory, phase change memory, spin-torque transfer memory.**

I. Introduction

Non-volatile, solid-state memories (NVMs) are poised to revolutionize storage in systems for high-performance computing. Flash memory is already finding applications in large-scale systems, and emerging technologies, such as phase-change memories (PCM), spin-torque transfer memories (STTM), and more exotic technologies (e.g., the memristor and carbon nanotube based memories) will provide orders of magnitude better performance than either conventional disks or flash-based solid-state drives can deliver. What impact these emerging technologies will have on future systems is not yet clear, but they will likely be more disruptive than flash memory has been (and continues to be).

The painfully slow performance of non-volatile storage has been an unfortunate reality for system designers for several decades. Systems designers have gone to great lengths to try to mitigate this poor performance: Operating systems employ complex schedulers for IO, and most of the complexity in database management systems is in buffer management and query optimizations designed, in large part, to minimize IO.

Slow disks have also had a large impact on how we build supercomputers. Many large-scale scientific applications benefit as much (or more) from the terabytes of DRAM that high-end systems provide as they do from the number of FLOPS. Using DRAM to provide support for large working sets has been the only practical solution, but it is expensive and energy-intensive.

Non-volatile, solid-state storage technologies promise to resolve these problems and enable high-performance systems that are faster, cheaper, and more agile than those we build today. Whether they can deliver on this promise remains to be seen, but it will certainly require that we understand the performance potential of these memories, their limitations, and how they will change the balance points within a system. It will also require evaluating the memories in the context of complete systems, since radically altering the cost of IO will reveal or create bottlenecks elsewhere.

This paper presents a comparison of memory technologies ranging from hard disks connected via SATA to advanced phase-change memories attached to a DDR3 memory bus. We evaluate the impact of these technologies on applications that vary widely in their bandwidth requirements, latency needs, and access patterns. We use these applications to identify several bottlenecks that these technologies reveal and to delineate where these technologies will have the most impact.

Since our study covers a range of NVM technologies, it compliments prior work that has focused on flash memory and flash-based SSDs. These include studies of the basic performance properties of SSDs [8] as well as their system-level performance for particular domains such as scientific [28], [33], [2], data center/database applications [24], [23], [32], and E-business [21], [25]. Emerging technologies have also sparked interest in their usefulness in building so-called ExaScale systems [1].

Our results demonstrate that advanced non-volatile technologies can provide large gains in both raw IO and application-level performance. In some cases, memories such as PCM and STTM can accelerate database applications by over 60× relative to disk, reducing the running time of scientifically important queries from days to hours. Finally, we analyze the performance of our system and show that higher levels of performance are possible, but that achieving them will require significant changes to operating

SC10 November 2010, New Orleans, Louisiana, USA 978-1- 4244-7558-2/10/$26.00

TABLE I
MEMORY TECHNOLOGY SUMMARY

Technology	Density		Latency	
	Cur.	Pred. [15]	Read	Write
SLC Flash	4 F^2	4 F^2	25 μs	200 μs
PCM	10 F^2	4 F^2	67.5 ns	215 ns
STTM	64 F^2	15 F^2	29.5 ns	95 ns
DRAM	6 F^2	4 F^2	25 ns	25 ns

systems and system architecture. These technologies also appear well-suited for paging in some applications: Paging to our prototype NVM storage system increases performance by 35× on average and can bring execution time to within a factor of 4 relative to running in DRAM without paging.

The rest of the paper is organized as follows: Section II describes the non-volatile memories we consider in this study. Sections III and IV describe the storage architectures and workloads we consider. Section V presents the results of our experiments and discusses their implications. Finally, in Section VI, we conclude.

II. NON-VOLATILE MEMORIES

Non-volatile storage technologies exhibit a range of performance and density characteristics that determine their impact on systems. Disks are, of course, well known and represent the *status quo* for non-volatile storage. Table I describes the solid state technologies we will consider in this study. Density predictions are from the International Technology Roadmap for Semiconductors (ITRS) [15] for 2015. Latencies for PCM and STTM are from [34] and [4], while DRAM and flash values are taken from typical datasheets. In addition to flash memory, which has entered wide use, we consider two advanced non-volatile memories: phase change memory (PCM) and spin-torque transfer memory (STTM). PCM-based products are already available and STTM devices should be on the market within a few years. We include DRAM to put these technologies in context.

The table lists the density of current prototype devices and the 2015 target densities for the devices from the ITRS. The densities are given in terms of F^2 where F is the minimum feature size of a given silicon manufacturing generation. This provides a metric for memory density that is independent of silicon manufacturing technology. For instance, the data show that PCM density is expected to increase by 2.5× by 2015 in addition to the increases offered by raw improvements in silicon manufacturing.

The ITRS values represent goals that the semiconductor manufacturing industry has set. Recent concerns about the continued scaling of both DRAM and NAND flash mean that PCM and/or STTM could surpass the density of these technologies in the future. PCM and STTM also consume much less power than DRAM when idle because they do not require refresh.

The latencies in the table for PCM, STTM, DRAM, and flash include just the latency to access the memory itself. They exclude the additional latency that busses and memory controllers may add.

While we focus on two emerging technologies (PCM and STTM) and describe them in more detail below, our study does not rely on specific characteristics of either memory. Our results will hold for any non-volatile, solid-state technology (e.g., the memristor or carbon nanotube memories) that presents a memory-like interface. From this perspective the main difference between technologies is read and write latency. We analyze the effect of device latency in detail in Section V-A.

A. Phase-change memory

PCM devices are already commercially available and research suggests they may play multiple roles in future systems. PCM stores data in the crystalline state of a chalcogenide metal layer [7]. As features sizes drop, PCM memories become faster and more energy efficient without sacrificing reliability. Recent work ([22], [30]) has demonstrated that this scalability will make PCM a viable main memory technology as DRAM's scaling begins to falter. NAND flash faces similar scaling challenges, making PCM a potentially attractive storage technology as well. The analysis in [22] provides a good characterization of PCM's performance and power consumption characteristics. We use the values from that work to model PCM devices.

Despite this promise, PCM does suffer from some reliability concerns: Like flash memory, it eventually wears out, although its lifetime (in terms of write cycles) is approximately 1,000× that of flash. As a result, PCM requires some form of wear management to ensure reasonable device lifetime. Recent research [11], [22], [37], [9], [29] has demonstrated that providing transparent wear-leveling for PCM is possible with minimal overhead. We use the start-gap scheme in [29] which provides wear-leveling with less that 1% overhead.

B. Spin-torque transfer memory

Discrete STTM memories will be available in the next 2-3 years, and their speed may eventually rival that of DRAM. STTM store bits as a magnetic orientation of one layer in a magnetic tunnel junction (MTJ). Depending on the orientation, the junction's resistance is either low (the "anti-parallel" state) or high (the "parallel" state) [12]. In this respect, STTM is similar to previous magnetic RAM technologies. STTM differs in how it sets the orientation in the MTJ: Instead of using electric fields as previous MRAM technologies have, STTM uses a current of polarized electrons. This avoids the scaling limitations that plagued field-based devices.

Currently, several companies including Grandis, Sony [14], Hitachi [19] and Renesas [35] have developed STTM prototypes. Cell sizes range from 48 to 64F^2 [35], [20]. Eventually STTM's density, latency, and power consumption may approach those of DRAM. In this work we base our estimates for performance on a published paper [34] and discussions with industry.

III. HARDWARE

Our testbed systems include multiple non-volatile storage technologies. Figure 1 shows the system configuration and

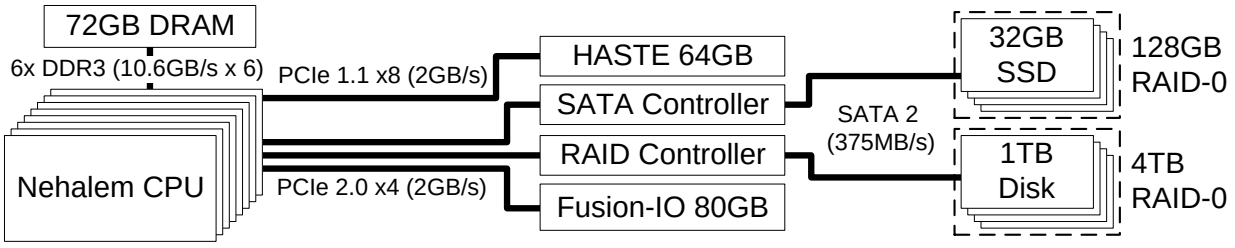

Fig. 1. **Test system** The test system incorporates many currently-available technologies along with a large amount of DRAM connected via multiple interconnects. PCIe links are full duplex, and the bandwidth values are per-direction.

TABLE II
TECHNOLOGIES UNDER TEST

Name	Capacity	Description
PCIe-attached PCM/STTM	64 GB	Modeled PCM or STTM attached via PCIe bus
DDR-attached PCM/STTM	64 GB	Modeled PCM or STTM attached via DDR3 DRAM bus
Fusion-IO	80 GB	Fusion-IO 80 GB PCIe SSD
RAID-SSD	128 GB	RAID-0 of 4x 32 GB X-25E SSDs
RAID-Disk	4 TB	RAID-0 of 4x 1 TB 7200 rpm hard drives

how each storage technology connects to the system. The testbed machines are two-socket, Core i7 Quad (a total of 8 cores) machines running at 2.26 GHz with 72 GB of physical DRAM and two 8 MB L2 caches (one per socket). The machines include a four disk RAID array of conventional 1TB hard drives, a four disk array of 32 GB Intel Extreme flash-based SSDs, and an 80 GB Fusion-IO PCIe-based solid-state disk. The final system component is the *High-performance Advanced Storage Technology Emulator*, or HASTE. HASTE contains four FPGAs that manage 64 GB of DRAM. We use the system DRAM and the DRAM in HASTE to emulate the emerging NVM technologies described in Section II. Table II lists the specifics of the storage devices we study. The following sections describe each device in detail.

A. HASTE: Emulating NVMs on the PCIe bus

HASTE can model PCIe-based SSDs that use advanced solid-state memories to store data. HASTE holds 64 GB of 667 MHz DDR2 DRAM, running at 250 MHz DDR (500M transfers per second), under the control of four Xilinx Virtex 5 FPGAs. It connects to the main system via an 8x PCIe 1.1 link with a peak bandwidth of 4 GB/s (2 GB/s in each direction). HASTE is based on the BEE3 FPGA prototyping system designed by Microsoft Research for use in the RAMP project [31] and sold by BEECube.

Internally, HASTE uses eight independent, high-performance memory controllers to access memory. A 4.5 GB/s ring-based network connects the controllers to provide uniform access to all 64 GB of storage. Figure 2 shows the internal architecture of HASTE.

One FPGA in HASTE has a PCIe link to the host system. This FPGA contains the request processor and handles all of the scheduling of accesses to the memory on all four FPGAs. The request processor handles requests that contain a host memory DMA address, sector number, length, operation (read or write), and tag. The sector number identifies which block of memory within HASTE to access, while the DMA address identifies a buffer in the host's DRAM. The scheduler can track up to 64 outstanding requests.

A DMA engine moves data between host system memory and a set of four local buffers. The request processor issues reads and writes to the eight DDR2 controllers over a token-based ring network with a peak bandwidth of 4.5 GB/s and round trip latency of 88 ns.

The request scheduler processes requests in order. For a write request, the DMA engine transfers data from the host's memory into a local buffer on the FPGA. When the target memory controller is idle, data streams over the ring network into a FIFO in the DDR controller before the DDR controller commits it to memory. For read requests, a similar process happens, but in reverse, with data moving from the memory controller to the DMA engine and then to host memory. When the data transfer completes, the scheduler raises an interrupt and sets a tag status bit in the hardware. The operating system completes the request once the interrupt arrives by checking the tag register.

To model PCM and STTM memories, we assume they have an internal architecture similar to DRAM chips. To access data, the memory controller issues a row address to all the chips on one DIMM. This "opens" the row for reading or writing and transfers its contents into a set of buffers in the memory chips. The aggregate size of the row across the chips is 8 KB, and once the row is open, accesses to that data can proceed at the memory's 250 MHz DDR bus speed (4.8 GB/s for 64-bit reads or writes).

To add the additional latency that PCM and STTM would incur, we modify the memory controller to add latency between the read address strobe and column address strobe commands during reads and extends the pre-charge latency after a write. We can adjust the extra delay independently for reads and writes in 4 ns increments. An interesting consequence of this arrangement is that HASTE only incurs the extra delay once when reading a 4 KB page.

To achieve high performance, HASTE requires significant changes to the Linux IO scheduler. Under normal operation, the thread requesting an IO operation places a request in a queue. A separate thread later removes the request and issues

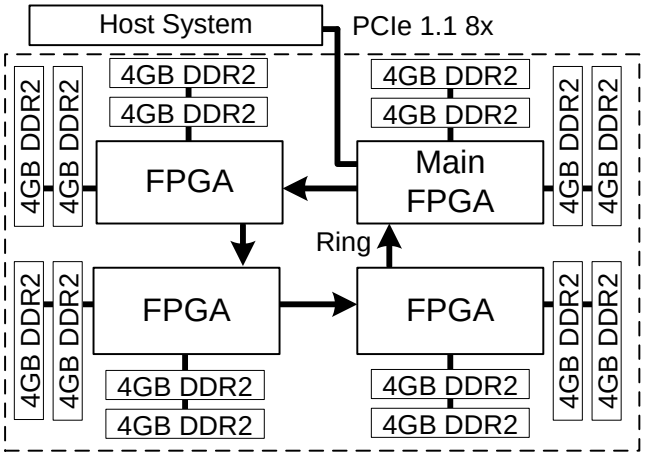

Fig. 2. **The HASTE system** A single 8x PCIe 1.1 endpoint connects the four FPGAs to the host system. A ring-based interconnect provides uniform latency access to all of the eight memory banks.

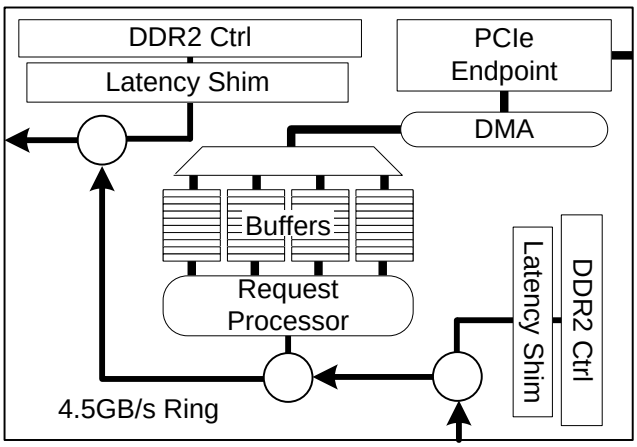

Fig. 3. **The HASTE controller** Two state machines manage requests for data transfers between the host memory, buffers on the FPGAs, and the DDR2 controllers distributed around the ring.

it to the storage device. This process adds at least 5 μs to the request latency. For disks, this latency is negligible, but for HASTE this cost is unacceptable.

The HASTE driver removes the queue entirely. The thread making the request issues it to the HASTE hardware and spins until it completes. The combination of these two changes reduces the latency for a single 4 KB access on HASTE from 23 to 16 μs. We examine the latency of HASTE accesses in more detail in Section V.

B. RAM-disks and DDR-attached NVMs

Modeling NVM storage attached to the processor's DDR memory bus also requires accounting for increased memory access times. We use a customized version of the Linux ramdisk driver that uses a large amount of kernel memory to implement a block device. The driver inserts extra delay on accesses to match the latency of non-volatile memories. We model the same delays described above for HASTE. Setting the delays to zero gives a ramdisk that runs at full DRAM speed.

C. Fusion-IO

The Fusion-IO card represents a significant step in SSD evolution. Instead of relying on conventional hard drive interfaces, it connects 25 high-performance single-level cell (SLC) NAND flash memory devices to the PCIe bus via an FPGA-based controller. It uses a custom driver that performs sophisticated scheduling and buffering using system DRAM. The driver consumes 800 MB of kernel memory for this purpose. The company rates the 80 GB card at 750 MB/s for reads and 500 MB/s for writes, with a read latency of 26 μs.

The Fusion-IO drive has default settings that are well tuned for both sequential and random accesses. Fusion-IO recommends using direct IO to bypass the file buffer cache whenever possible to maximize performance. We following this recommendation.

D. RAID-SSD

The SSD-based RAID-0 array in our system contains four Intel Extreme 32 GB SSDs (SSDSA2SH032G1GN). Intel rates the SSDs at 250 MB/s for reads and 170 MB/s for writes giving a theoretical peak bandwidth of 1 GB/s for four drives. The drives have a nominal latency of 75 μs.

Our measurements and other recent work [13] shows that software RAID provides better performance for SSD-based arrays than hardware controllers, because the processors on hardware RAID controllers become a bottleneck. Therefore, we use software RAID for this array. We tuned the array separately for sequential and random IO operations (using XDD), and found the same settings (64 KB stripe size) were optimal in both cases. We use this configuration in all our experiments.

E. RAID-Disk

The disk-based RAID-0 array in our system contains four 1TB Hitachi HDE721010SLA330 drives that spin at 7200rpm. They attach to an 8-channel 3ware 9650SE-8LPML RAID controller that can provide 256MB of on-board DRAM for caching and write buffering. We tuned the array for both sequential and random workloads. For sequential accesses, a stripe size of 128KB with no write journaling was optimal. For random accesses, a stripe size of 16KB with no write journaling achieved the highest performance.

IV. Workloads

Solid-state non-volatile memories will potentially find use in many different types of applications, and their impact will vary depending on how systems use them. There are at least three large categories of applications that may benefit significantly:

1) **Raw device and file access** In these applications, NVMs replace disks as the primary storage medium. Applications access the data via normal file operations

TABLE III
BENCHMARKS AND APPLICATIONS

Name	Data footprint	Description
IO benchmarks		
XDD Sequential	55 GB	4 MB sequential reads, writes, or reads/writes from 16 threads
XDD Random	55 GB	4 KB random reads, writes, or reads/writes from 16 threads
Linux Build	0.5 GB	Compilation of the Linux 2.6 kernel
Linux Patch	17 GB	Applies patches to the Linux kernel from version 2.6.0 to 2.6.29
Postmark	0.5 GB	Models an email server
Database applications		
Berkeley-DB Btree	16 GB	Transactional updates to a B+tree key/value store
Berkeley-DB HashTable	16 GB	Transactional updates to a hash table key/value store
BiologicalNetworks	35 GB	Biological database queried for properties of genes and biological-networks
PTF	50 GB	Palomar Transient Factory database real time sky survey queries
Memory-hungry applications		
Thrash	4-60 GB	Randomly update values in a large array.
DGEMM	21 GB	Matrix multiplication and addition with 30,000×30,000 matrices
BT	11 GB	Computational fluid dynamics simulation
CG	18 GB	Computes an approximation of the smallest eigenvalue of a matrix
IS	35 GB	Sorts integers with the bucket sort algorithm
LU	9 GB	LU matrix decomposition
MG	28 GB	Solves three-dimensional matrices with the multigrid method
SP	12 GB	Simulated CFD code solves scalar-pentadiagonal bands of linear equations
UA	8 GB	Solves a heat transfer problem on an unstructured, adaptive grid

(`open()`, `close()`, `read()`, `write()`, etc.) or by accessing the raw block device directly.

2) **Database applications** Databases are playing a growing role in many scientific applications. They provide sophisticated buffer management systems meant to hide the latency of slow disks. Buffer management and file system efficiency both impact performance.
3) **Paging** Using non-volatile storage to virtualize DRAM can increase effective memory capacity. The impact of paging on application performance is potentially quite large, especially for hard drive-based paging systems. Solid-state storage technologies, however, may be fast enough to make paging a useful alternative to increasing DRAM capacity in high-performance systems.

Table III summarizes the applications we use in this study, and we describe them below in more detail. For all applications that require a file system, we use XFS. Section V uses these workloads to evaluate NVM performance.

A. Raw device and file access

We use four different applications to measure basic device and file performance.

XDD

XDD [36] characterizes basic IO bandwidth and latency performance. We use XDD to perform 12 tests with the following characteristics: 100% reads, 100% writes, and 50% reads/writes; sequential accesses of 4 MB chunks and random accesses of 4 KB chunks; and with and without a file system. We use 16 threads in each test.

Linux Build

The build workload compiles version 2.6.23.1 of the Linux kernel source tree. All options are enabled in the configuration to maximize the amount of work done by benchmark. Build does file IO but is compute bound.

Linux Patch

Patch applies patches to the Linux kernel from version 2.6.0 to 2.6.29. It uncompresses each patch and applies it to file throughout the source tree.

Postmark

Postmark [18] is a file IO benchmark that emulates the activity of a large email server. The benchmark works on a pool of 10,000 files ranging in size from 1 KB to 64 MB, and performs 100,000 transactions, each consisting of a pair of read or write and create or delete operations. We modified Postmark to allow it use direct IO to bypass the system buffer cache. For our tests, we ran it with and without direct IO and report the best value.

B. Database applications

To measure basic database operation and transaction processing performance, we use BerkeleyDB. We use two full-fledged scientific databases to measure application-level performance.

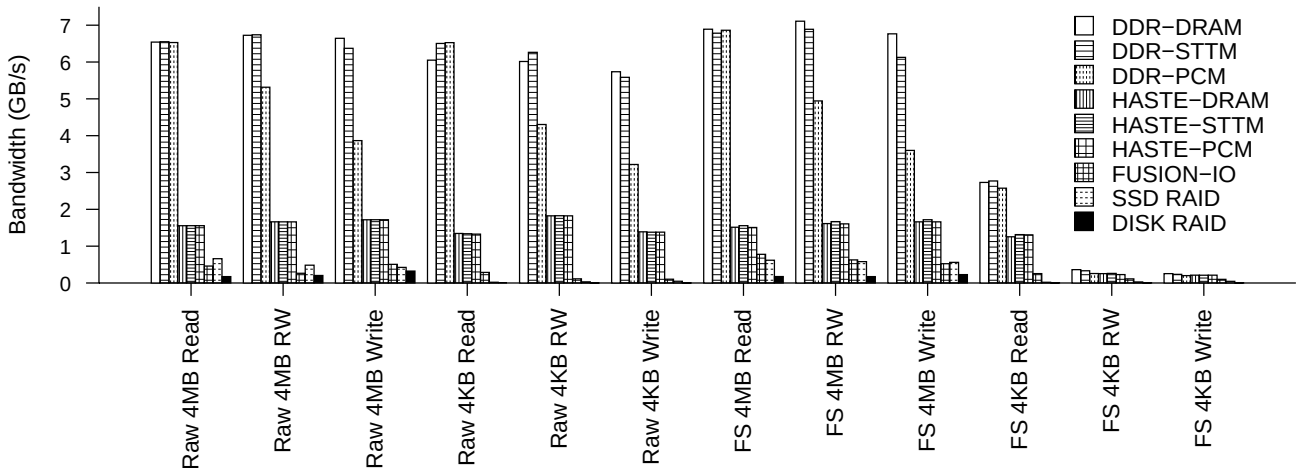

Fig. 4. **Device bandwidth** Bandwidth across different storage technologies differs significantly with access type and whether a file system is present.

Berkeley DB

Berkeley Database (Berkeley DB) is a popular high-performance embedded database and it serves as a generic key/value store for a variety of applications. This workload performs random inserts and deletes in a 16 GB key/value store implemented as either a B+tree or a hash table. All updates to storage are done through ACID transactions.

Biological pathway analysis

BiologicalNetworks [5] is a systems biology software platform for analysis and visualization of biological pathways, gene regulation, and protein interaction networks. Typical usage performs a large number of long and short-running queries to a PostgreSQL database. These queries are a bottleneck for researchers in this domain when they have to analyze pathways using a visual interface. Our tests include a series of real-world BiologicalNetworks queries over a database sized to fit within our storage systems.

Palomar Transient Factory

The Palomar Transient Factory (PTF) [27] uses several large databases of astronomical data to classify objects that appear suddenly in the night sky (i.e., "transients"). PTF typically identifies on the order of 100 new transients every minute it is in operation along with 1000 spurious detections related to image artifacts, etc. The queries vet and classify the transients in order to quickly schedule more detailed observations very quickly (e.g., in less than 24 hours or even in real time), so query response times are critical. Our workload runs six of the most time critical queries on a 50 GB database.

C. Paging applications

Some high-performance applications running on supercomputers benefit as much or more from the large DRAM capacities that the machines offer as they do from FLOPS. Using non-volatile storage as virtual memory can effectively increase memory capacity, but the poor performance of disk-based storage leads to unacceptable performance degradation. If a solid-state storage array is fast enough, it may alleviate this problem and make it possible to run large memory applications on smaller, more efficient machines. We use several applications to measure paging performance on our solid-state storage technologies.

Thrash

Thrash is a simple paging microbenchmark that allocates a large array of integers, touches each page once, and then spawns 16 threads that randomly update entries in the array and perform no other work. To avoid contention effects, each thread accesses a separate region of the array. We vary the data set size between 4 GB (which will fit in DRAM) and 60 GB.

DGEMM

DGEMM performs double-precision matrix multiplication. Our implementation uses GotoBLAS2 [6] and operates on 30,000×30,000 element matrices.

NAS Parallel Benchmarks

We use applications from the NAS Parallel Benchmark (NPB) suite [3] version 3.3 written for OpenMP ([26], [16]). We use the BT, CG, IS, LU, MG, SP, and UA kernels running with class D problem sizes because they have large data sets (8 to 35 GB) that force the system to page. We run each benchmark with 16 threads.

V. Results

This section evaluates our storage arrays using the benchmarks described in the previous section.

A. File and raw device access

XDD measures operation latency and aggregate bandwidth and can quantify the impact of the file system on performance. It also demonstrates that the impact of non-volatile storage technology parameters varies depending on system architecture.

Bandwidth and latency

Figure 4 shows the average bandwidth of each storage technology measured using XDD running 16 threads.

The data show the decrease in performance as bus bandwidth shrinks and device latencies increase. DDR-DRAM's 7.1 GB/s peak performance dwarfs all of the other storage technologies because of the disparity in bus bandwidth between the 6 DDR3 channels (63 GB/s total) and the PCIe links (2 GB/s). DDR-STTM adds 70 ns of latency to each write compared to DDR-DRAM with a resulting performance drop of 4%, while DDR-PCM's 120 ns of additional latency vs. DDR-DRAM causes a 42% drop in bandwidth. The costs of the system calls, file system, and operating system are steep: They prevent the ramdisk from utilizing more than 12% of the bandwidth that the DDR3 memory bus can deliver.

HASTE-DRAM, HASTE-STTM, and HASTE-PCM all achieve nearly the same performance on 4MB accesses, reaching 1.5 GB/s for reads and 1.7 GB/s for writes. Write performance is higher because HASTE can make more efficient use of buffers during write requests.

Request overhead makes up a larger percentage of the request latency for 4 KB writes, limiting bandwidth to 1.33 GB/s for reads and 1.37 GB/s for writes. Variation in system performance between runs accounts for the small variations in bandwidth across memory types.

The XFS filesystem adds significant overheads in some cases. For instance, for HASTE-DRAM, it reduces performance by 50 MB/s for both 4 MB reads and 4 MB writes, while 4 KB writes suffer a 84% drop in performance.

The long latency of flash memory limits Fusion-IO's peak performance on 4 KB accesses (280 MB/s), but its customized architecture delivers nearly 5× more read bandwidth than the SSD array, despite the fact that the SSD array contains 55 more flash devices (80 vs. 25). We suspect this gap is due to Fusion-IO's lower per-operation latency and its more streamlined architecture. Fusion-IO's 4KB random writes are 3.3× better than the SSD. Interestingly, the SSD array has 43% better sequential read performance than the Fusion-IO drive. Sequential writes are almost the same across both drives, suggesting that the additional parallelism in the SSD array enables slightly better read bandwidth, but write latency limits performance on both devices.

Figure 5 measures latency for read and write operations with and without the file system for each storage technology. Note the logarithmic scale. We collected these data with XDD running a single thread and performing random 4 KB accesses.

Without a file system, latency is similar for all three HASTE configurations (DRAM, STTM, and PCM) and for both reads and writes ($\sim$15μs). The DDR-attached NVM has lower, but also more variable latency: DRAM accesses take 3μs for both reads and writes, but the extra latency for PCM and STTM slow down accesses considerably, especially for writes. We explore these effects in more detail below.

File system overheads

Figures 4 and 5 both show that the file system has a large impact on both bandwidth and latency, and that the impact is much larger for faster devices. For DISK RAID, XFS has relatively little impact: It never reduces bandwidth by more than 10%. As the storage systems get faster, though, the impact increases. SSD RAID sees a reduction of between 5% (sequential reads and writes) and 10% (random reads and writes) in bandwidth and a 4% increase in latency. For HASTE, random write bandwidth drops from 1.3 GB/s to 210 MB/s. Finally, for the ramdisk, the file system increases latency by 2.7× and reduces random access bandwidth by 96% for writes and 55% for reads.

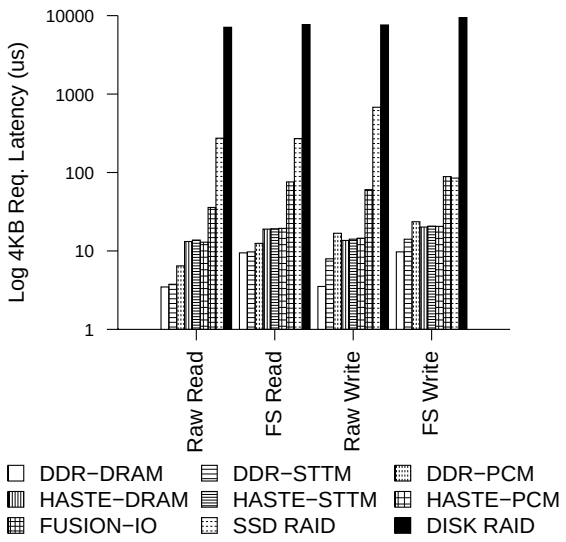

Fig. 5. **Device latency** The differences in latency between devices and interconnects are very large. NVMs such as PCM and STTM can offer between two and three orders of magnitude reduction in latency compared to disk.

The latency cost of the file system is also large. For the DDR-attached and HASTE devices, the file system consistently increases per-access latency by 6 μs. This amounts to an increase of 93% and 50% for the DDR-attached PCM configurations and HASTE-PCM, respectively. For DISK RAID the increase is larger in absolute terms (580 μs), but is a much smaller percentage (8%).

For comparison, we ran the same experiments with ext3 instead of XFS. The two file systems had almost identical effects on DISK RAID performance, but for all of the faster storage devices ext3 reduced performance much further. This was especially true for sequential accesses on fast devices: Adding ext3 reduced bandwidth for those accesses by 74%, while XFS actually increased bandwidth by 2%.

These file system overheads are representative of the challenges that fast non-volatile storage arrays present: System designers have assumed that IO devices are slow and that assumption permeates the entire system. Optimizations in HASTE and Fusion-IO drivers eliminated many of these costs at the block device level, but file systems clearly require additional effort. We expect that IO intensive application will also implicitly assume that IO is slow and will require optimization to take full advantage of fast storage.

This work has begun at the file system level: For instance, DFS [17] is a file system optimized specifically for Fusion-IO-style drives. BPFS [10] targets fast, byte-addressable memories.

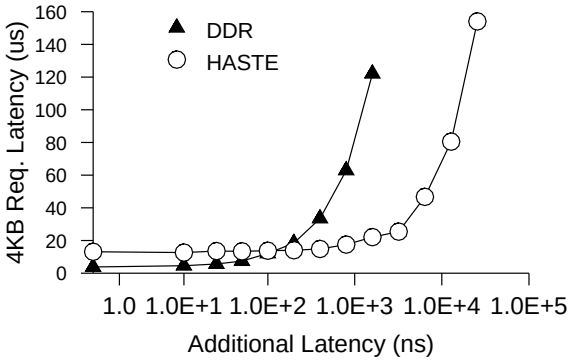

Fig. 6. **The impact of NVM latency** Increasing the latency of the raw memory devices has a larger impact for DDR configurations than for HASTE, because HASTE's memory controller perform a complete IO request at once rather than using the CPU to copy data.

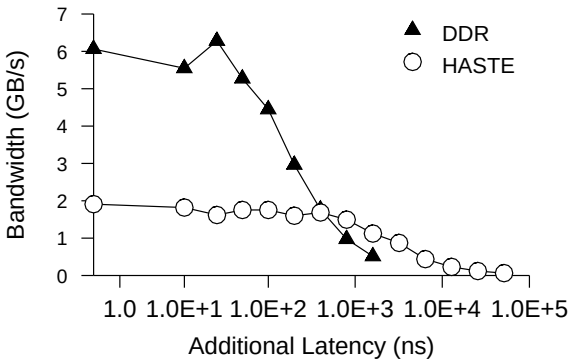

Fig. 7. **NVM latency and bandwidth** Memory latency has a smaller impact on total latency for HASTE because HASTE has greater internal parallelism that it uses to hide latency.

Memory technology performance impact

Figures 6 and 7 explore the impact of increased memory latency in more detail. In both figures, we vary the latency for reads and writes between 25 ns and 51,200 ns (about twice the read latency of SLC NAND flash), and measure the latency without a file system. For the DDR-attached NVMs, we show data only to 1600 ns, since it is likely that, for slower memories, attaching them to the DDR bus is not worthwhile.

The data in Figure 6 show that increased memory latency has a much larger impact for the DDR-attached memories than for HASTE. The reason for this difference stems from differences in how the ramdisk and HASTE access memory. In HASTE the operating system issues a DMA request to the HASTE hardware. For HASTE-DRAM, servicing the request takes about 6.4 μs. Of this, 6μs is PCIe transfer time and interrupt processing and about 125 ns is due to the interconnect and buffering within HASTE. The memory access accounts for just 280 ns. The remainder of the 16 μs total access latency (as seen by XDD) is in the operating system and includes several lock acquisitions and a context switch. With all this overhead, the memory access time accounts for just 4.5% of total operation latency while the operating system accounts for 63%.

For DDR-attached memories, total access time is much smaller (just 3 μs). This is due both to the removal of the PCIe bus, but also to a simpler driver for the ramdisk: The ramdisk driver does not include the lock acquisition, context switch, or interrupt processing. This reduction in overhead translates to greater relative impact from increased NVM latency.

In addition, the HASTE memory controllers are fully dedicated to servicing one 4 KB request at a time, so they can stream the data out at the full speed of the DDR interface. It also means that HASTE only has to pay the high cost of accessing NVMs once per 4 KB access (as described in Section III).

In contrast, the accesses to the DDR-attached memories come from the processor. It must issue a long series of 64-bit reads or writes to the memory system. These accesses must traverse the cache hierarchy and compete with other requests for access to the DRAM controllers and to the 6 DRAM busses in the system (vs. the 8 in HASTE). As a result, if raw memory latency exceeds 800 ns, the DDR-attached memory's latency exceeds that of HASTE.

Increased latency impacts HASTE's bandwidth less than it affects DDR-attached configurations. The reason is the lock acquisitions in the kernel combined with HASTE's ability to use parallelism to hide latency. The lock protects the HASTE driver's internal control structures, and prevents the OS from issuing more than one operation every 5 μs, on average. Contention for both this lock and the PCIe bus means there is little parallelism in the HASTE hardware when the NVM is fast. As operation latency increases, the number of outstanding requests rises. HASTE contains eight DDR memory controllers which can all access memory in parallel. HASTE's bandwidth does not begin to drop until all these controllers are consistently busy. In our system this occurs for latencies larger than 12.8 μs.

The caching and memory bus contention effects described above also exacerbate the impact of increased latency on the DDR configurations' bandwidth.

File-intensive application performance

File system overheads limit the performance of Build, Patch, and Postmark, our file-intensive applications, and much of this overhead is due to the buffer cache. Moving from disk to a faster storage device with better random access performance helps, but the choice of which storage device seems to make little difference. For Patch, we see a nearly uniform 4$\times$ speedup for HASTE and the DDR memories compared to DISK RAID. For Postmark, there is more variation: We see improvements high as 3.3$\times$ for the DDR devices, 3.5$\times$ for HASTE and 2.9$\times$ for FUSION-IO over DISK RAID. Interestingly, for Postmark and Patch, using direct IO improves performance for all storage devices. Build is a compute bound benchmark, and the latency of the storage technology is mostly hidden: Moving to NVMs improves performance by only 10% on average.

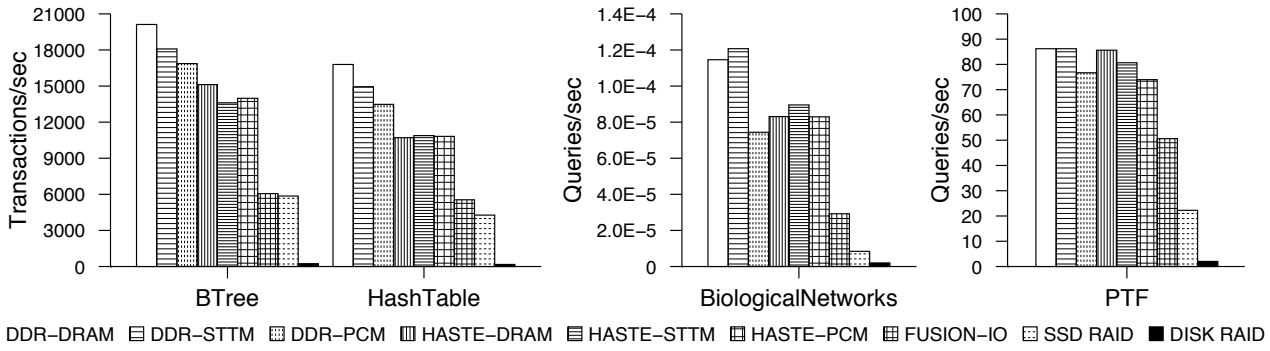

Fig. 8. **Database application performance** The graph on the left shows the throughput in transactions/second of a B+tree and hash table implemented in BerkeleyDB. The center and right graphs display the queries/sec for the BiologicalNetworks and PTF databases respectively. BiologicalNetworks runs a very long query, and in the case of DISK RAID, we had to stop its execution after 140 hours.

B. Database applications

Figure 8 measures the performance of our database applications. The left graph in the figure shows transactions per second for Btree and HashTable running on BerkeleyDB. The DDR and HASTE NVM arrays improve throughput for BerkeleyDB by between 58 and 97× over DISK RAID. The throughput increases for the BiologicalNetworks (center) and PTF (right) databases are lower, and although still large, the benefits of NVMs on the DDR bus versus HASTE drop significantly, especially for the PCM configuration. We suspect this is due to the greater complexity and correspondingly larger overheads of the PostgreSQL database compared to BerkeleyDB. As with the file system, optimizing the database software layer may expose more of the underlying hardware performance to the application.

Despite those overheads, the BiologicalNetworks and PTF results provide excellent case studies for the large practical benefits that advanced non-volatile memories can offer.

For the BiologicalNetworks, the largest query in our workload ran for over 140 hours on RAID-disk without completing and took over 33 hours to complete on the SSD RAID. Fusion-IO reduces the running time for this query to under 10 hours (2.9E-05 queries/s). HASTE improves performance by an additional 3×, with all three versions achieving similar performance. The ramdisk reduces runtime by a further 36% (to 145 minutes or 1.1E-04 queries/s) for STTM and DRAM, while the PCM version actually slows down slightly relative to HASTE.

For PTF, moving to non-volatile memory dramatically increases performance. The SSD array achieves one fourth the throughput of HASTE or NVMs on the DDR memory bus. Disk is even worse with 40× fewer queries per second. Using HASTE or the ramdisk makes it possible to process queries in a little over 1 ms on average. This is especially significant because it would allow the PTF to categorize transients in real time as they appear.

C. Paging applications

Many important scientific applications have large memory working sets and limited parallelism. To achieve reasonable performance, these jobs run on large-scale supercomputers because of their large DRAM capacity, but significantly underutilize the machine's computational resources. The result is that the applications both increase contention for these large machines but also run with very low energy efficiency, since they incur the energy cost of mostly-idle compute nodes.

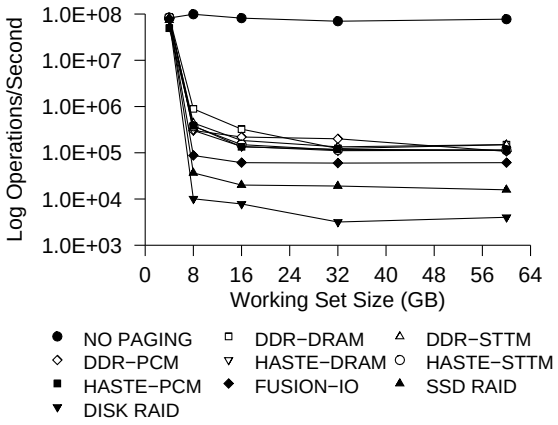

Fig. 9. **Paging microbenchmark performance** Although paging reduces performance dramatically, paging to advanced NVMs offers between one and two orders of magnitude improvement in performance. Performance for all HASTE configurations level out at 1×10^5.

If NVMs could serve as a backing store for paged virtual memory without crippling performance, they could increase the efficiency of these computations and reduce contention for supercomputers that have both large compute and large memory capabilities. This approach is one of several motivations for building a machine called Flash Gordon at the San Diego Supercomputing Center that will incorporate several terabytes of flash memory.

Figure 9 contains the results for our paging microbenchmark, thrash. The top line is the performance running with 64GB of DRAM, so very little paging occurs. The lines show the performance (in random updates per second) for each memory technology as the working set size increases.

The figures make it clear why spinning disks are not use-

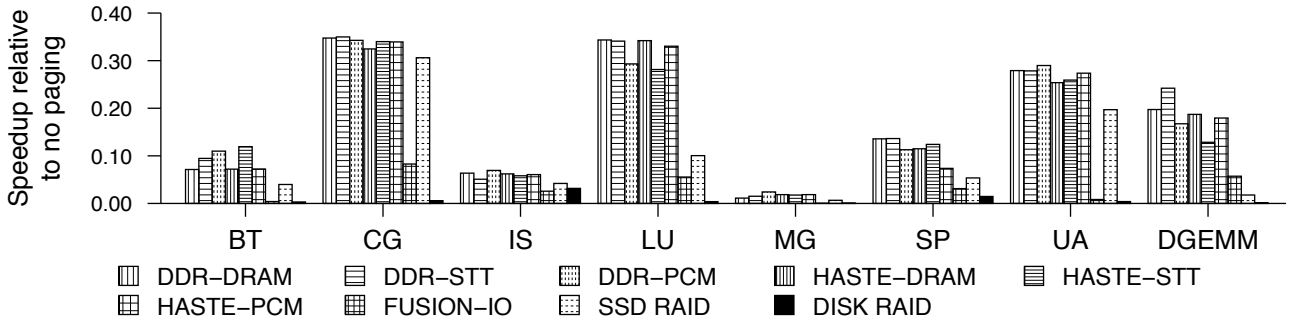

Fig. 10. **Paging applications** NVMs can significantly reduce the cost of the paging for memory-intensive applications. This makes paging a viable option for expanding working sets in some cases.

ful for paging: Paging to DISK RAID reduces performance by up to 20,000×. SSD RAID and Fusion-IO reduce that margin to around 1000-4000×. Moving to PCM or STTM on either the DDR bus or in HASTE closes the gap to just 516-683×. If these slow downs hold for real applications, intensive paging would not be a feasible option on any technology.

Figure 10 shows the impact for real applications is much smaller. The applications require between 8 and 35 GB of memory, but we limit the applications to just 8 GB of DRAM, forcing them to page. For comparison, we also run the workloads with sufficient DRAM to prevent paging, and present performance relative to the no paging version. The graph measures performance in application instructions per second collected via hardware performance counters, since the applications would take several days or weeks to complete when paging to RAID-disk. We start measurements once the applications have finished their initialization phase.

Paging to spinning disk results in very poor performance: It reduces performance by between 32 and 1515×. SSD RAID and Fusion-IO do better, but still reduce performance by an average of 11 and 33× respectively. HASTE slows down performance by only 5.8×, which might be acceptable for some applications. Paging to the DDR configurations reduces this gap to 5.5×, which is just 12% better than HASTE. Thus, using a high-performance storage device like HASTE for paging is a good way to increase the effective working set of an application.

The impact of paging varies, not surprisingly, with the memory requirements of the program. For instance, both IS and MG have large memory footprints (35 and 28 GB respectively) and little spatial locality, resulting in slow downs of 16 and 54× when paging to HASTE. In contrast, CG and LU use less memory (18 and 9 GB, respectively) and exhibit more spatial locality, so performance drops by 66-68%. UA has the smallest working set, but its unstructured accesses lead to larger slowdowns than LU and CG.

VI. Conclusion and future work

This paper has characterized the performance of currently available and emerging solid-state storage technologies both in terms of raw performance and application-level impact.

We find that NVMs offer large gains in latency and bandwidth and can significantly accelerate database applications, reducing query execution time from days to hours in some cases. Their usefulness as backing store for paged virtual memory varies between applications depending on paging frequency.

While NVMs provide large improvements in latency and bandwidth, performance still falls short of what these memory devices should be able to deliver. Our latency measurements provide a case in point: STTM and PCM chips will be able to perform a read operation in between 29 and 67 ns, yet the total latency for a DDR-attached PCM or STTM memory is at least 3 μs, roughly 100× longer. The transfer time for 4KB over our DDR3 memory bus is 360 ns, leaving 2.6 μs of pure overhead. The PCIe bus adds even more overhead.

The overheads stem from inefficiencies both in the hardware (e.g., PCIe latency) and software (e.g., operating and file systems overheads). HASTE removes several of these inefficiencies and represents a lower bound on the performance that PCIe-attached advanced NVMs can provide. We are still refining HASTE, its driver, and how it interacts with the OS and file system. We believe that further improvements are possible, but they may require more substantial changes to many parts of the system including the file system and applications. Understanding what these changes should be and integrating them elegantly into existing systems is the central challenge in fully exploiting fast non-volatile memories.

Acknowledgments

This work was sponsored in part by the National Science Foundation under NSF OCI #0951583 entitled “I/O Modeling EAGER”, by NSF OCI #0910847 entitled “Gordon: A Data Intensive Supercomputer,” and by hardware donations from Xilinx. The authors would also like to thank Nathan Goulding, Brett Kettering, and James Nunez.

References

[1] Exascale computing study: Technology challenges in achieving exascale systems. Technical Report TR-2008-13, University of Notre Dame, CSE Department, September 2008.

[2] R. Agrawal, A. Ailamaki, P. A. Bernstein, E. A. Brewer, M. J. Carey, S. Chaudhuri, A. Doan, D. Florescu, M. J. Franklin, H. Garcia-Molina, J. Gehrke, L. Gruenwald, L. M. Haas, A. Y. Halevy, J. M. Hellerstein, Y. E. Ioannidis, H. F. Korth, D. Kossmann, S. Madden, R. Magoulas, B. C. Ooi, T. O'Reilly, R. Ramakrishnan, S. Sarawagi, M. Stonebraker, A. S. Szalay, and G. Weikum. The claremont report on database research. *Communications of the ACM*, 52(6):56–65, 2009.

[3] D. H. Bailey, E. Barszcz, J. T. Barton, D. S. Browning, R. L. Carter, L. Dagum, R. A. Fatoohi, P. O. Frederickson, T. A. Lasinski, R. S. Schreiber, H. D. Simon, and V. Venkatakrishnan. The nas parallel benchmarks, 1994.

[4] F. Bedeschi, C. Resta, O. Khouri, E. Buda, L. Costa, M. Ferraro, F. Pellizzer, F. Ottogalli, A. Pirovano, M. Tosi, R. Bez, R. Gastaldi, and G. Casagrande. An 8mb demonstrator for high-density 1.8v phase-change memories. *VLSI Circuits, 2004. Digest of Technical Papers. 2004 Symposium on*, pages 442–445, June 2004.

[5] Biological networks website. http://biologicalnetworks.net/.

[6] Gotoblas2 website. http://www.tacc.utexas.edu/tacc-projects/.

[7] M. J. Breitwisch. Phase change memory. *Interconnect Technology Conference, 2008. IITC 2008. International*, pages 219–221, June 2008.

[8] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *SIGMETRICS '09: Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, pages 181–192, New York, NY, USA, 2009. ACM.

[9] S. Cho and H. Lee. Flip-n-write: A simple deterministic technique to improve pram write performance, energy and endurance. In *To appear in MICRO 2009*, 2009.

[10] J. Condit, E. B. Nightingale, E. Ipek, D. Burger, B. Lee, and D. Coetzee. Better i/o through byte-addressable, persistent memory. In *SOSP '09: Proceedings of the twenty-second ACM Symposium on Operating systems principles*. To appear.

[11] G. Dhiman, R. Ayoub, and T. Rosing. Pdram: a hybrid pram and dram main memory system. In *DAC '09: Proceedings of the 46th Annual Design Automation Conference*, pages 664–469, New York, NY, USA, 2009. ACM.

[12] B. Dieny, R. Sousa, G. Prenat, and U. Ebels. Spin-dependent phenomena and their implementation in spintronic devices. *VLSI Technology, Systems and Applications, 2008. VLSI-TSA 2008. International Symposium on*, pages 70–71, April 2008.

[13] J. He, A. Jagatheesan, S. Gupta, J. Bennett, and A. Snavely. Dash: A recipe for a flash-based data intensive supercomputer. November 2010.

[14] M. Hosomi, H. Yamagishi, T. Yamamoto, K. Bessho, Y. Higo, K. Yamane, H. Yamada, M. Shoji, H. Hachino, C. Fukumoto, H. Nagao, and H. Kano. A novel nonvolatile memory with spin torque transfer magnetization switching: spin-ram. *Electron Devices Meeting, 2005. IEDM Technical Digest. IEEE International*, pages 459–462, Dec. 2005.

[15] International technology roadmap for semiconductors: Emerging research devices, 2009.

[16] H. Jin, H. Jin, M. Frumkin, M. Frumkin, J. Yan, and J. Yan. The openmp implementation of nas parallel benchmarks and its performance. Technical report, NASA, 1999.

[17] W. K. Josephson, L. A. Bongo, D. Flynn, and K. Li. Dfs: A file system for virtualized flash storage. In *Proceedings of FAST 10: 8th USENIX Conference on File and Storage Technologies*, 2010.

[18] J. Katcher. Postmark filesystem performance benchmark. http://www.netapp.com/tech_library/3022.html.

[19] T. Kawahara, R. Takemura, K. Miura, J. Hayakawa, S. Ikeda, Y. Lee, R. Sasaki, Y. Goto, K. Ito, I. Meguro, F. Matsukura, H. Takahashi, H. Matsuoka, and H. Ohno. 2mb spin-transfer torque ram (spram) with bit-by-bit bidirectional current write and parallelizing-direction current read. *Solid-State Circuits Conference, 2007. ISSCC 2007. Digest of Technical Papers. IEEE International*, pages 480–617, Feb. 2007.

[20] T. Kawahara, R. Takemura, K. Miura, J. Hayakawa, S. Ikeda, Y. M. Lee, R. Sasaki, Y. Goto, K. Ito, T. Meguro, F. Matsukura, H. Takahashi, H. Matsuoka, and H. Ohno. 2 mb spram (spin-transfer torque ram) with bit-by-bit bi-directional current write and parallelizing-direction current read. *Solid-State Circuits, IEEE Journal of*, 43(1):109–120, Jan. 2008.

[21] D.-S. Ko and S.-K. Cheong. Web performance enhancement of e-business system using the ssd. In *FGCNS '08: Proceedings of the 2008 Second International Conference on Future Generation Communication and Networking Symposia*, pages 81–84, Washington, DC, USA, 2008. IEEE Computer Society.

[22] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting phase change memory as a scalable dram alternative. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 2–13, New York, NY, USA, 2009. ACM.

[23] S.-W. Lee, B. Moon, and C. Park. Advances in flash memory ssd technology for enterprise database applications. In *SIGMOD '09: Proceedings of the 35th SIGMOD international conference on Management of data*, pages 863–870, New York, NY, USA, 2009. ACM.

[24] S.-W. Lee, B. Moon, C. Park, J.-M. Kim, and S.-W. Kim. A case for flash memory ssd in enterprise database applications. In *SIGMOD '08: Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1075–1086, New York, NY, USA, 2008. ACM.

[25] D. Narayanan, E. Thereska, A. Donnelly, S. Elnikety, and A. Rowstron. Migrating server storage to ssds: analysis of tradeoffs. In *EuroSys '09: Proceedings of the 4th ACM European conference on Computer systems*, pages 145–158, New York, NY, USA, 2009. ACM.

[26] NASA. Nas parallel benchmarks, March 2010. http://www.nas.nasa.gov/Resources/Software/npb.html.

[27] Palomar transient factory (ptf) website. http://www.astro.caltech.edu/ptf/.

[28] S. Park and K. Shen. A performance evaluation of scientific i/o workloads on flash-based ssds. In *Workshop on Interfaces and Architectures for Scientific Data Storage*, 2009.

[29] M. K. Qureshi, J. Karidis, M. Franceschini, V. Srinivasan, L. Lastras, and B. Abali. Enhancing lifetime and security of pcm-based main memory with start-gap wear leveling. In *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*, pages 14–23, New York, NY, USA, 2009. ACM.

[30] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable high performance main memory system using phase-change memory technology. *International Symposium on Computer Architecture*, June 2009.

[31] The ramp project. http://ramp.eecs.berkeley.edu/index.php?index.

[32] K. Schmidt, Y. Ou, and T. Härder. The promise of solid state disks: increasing efficiency and reducing cost of dbms processing. In *C3S2E '09: Proceedings of the 2nd Canadian Conference on Computer Science and Software Engineering*, pages 35–41, New York, NY, USA, 2009. ACM.

[33] A. S. Szalay, G. C. Bell, H. H. Huang, A. Terzis, and A. White. Low-power amdahl-balanced blades for data intensive computing. *SIGOPS Oper. Syst. Rev.*, 44(1):71–75, 2010.

[34] R. Takemura, T. Kawahara, K. Miura, J. Hayakawa, S. Ikeda, Y. Lee, R. Sasaki, Y. Goto, K. Ito, T. Meguro, F. Matsukura, H. Takahashi, H. Matsuoka, and H. Ohno. 2mb spram design: Bi-directional current write and parallelizing-direction current read schemes based on spin-transfer torque switching. *Integrated Circuit Design and Technology, 2007. ICICDT '07. IEEE International Conference on*, pages 1–4, 30 2007-June 1 2007.

[35] H. Tanizaki, T. Tsuji, J. Otani, Y. Yamaguchi, Y. Murai, H. Furuta, S. Ueno, T. Oishi, M. Hayashikoshi, and H. Hidaka. A high-density and high-speed 1t-4mtj mram with voltage offset self-reference sensing scheme. *Solid-State Circuits Conference, 2006. ASSCC 2006. IEEE Asian*, pages 303–306, Nov. 2006.

[36] Xdd version 6.5. http://www.ioperformance.com/.

[37] P. Zhou, B. Zhao, J. Yang, and Y. Zhang. A durable and energy efficient main memory using phase change memory technology. In *ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture*, pages 14–23, New York, NY, USA, 2009. ACM.