AN INTEGRATED SIMULATION INFRASTRUCTURE FOR THE ENTIRE MEMORY HIERARCHY: CACHE, DRAM, NONVOLATILE MEMORY, AND DISK

Contributors

Jim Stevens
University of Maryland

Paul Tschirhart
University of Maryland

Mu-Tien Chang
University of Maryland

Ishwar Bhati
University of Maryland

Peter Enns
University of Maryland

James Greensky
Intel Labs

Zeshan Chisti
Intel Labs

Shih-Lien Lu
Intel Labs

Bruce Jacob
University of Maryland

As computer systems evolve towards exascale and attempt to meet new application requirements such as big data, conventional memory technologies and architectures are no longer adequate in terms of bandwidth, power, capacity, or resilience. In order to understand these problems and analyze potential solutions, an accurate simulation environment that captures all of the complex interactions of the modern computer system is essential. In this article, we present an integrated simulation infrastructure for the entire memory hierarchy, including the processor cache, the DRAM main memory system, and nonvolatile memory, whether it is integrated as hybrid main memory or as a solid state drive. The memory simulations we present are integrated into a full system simulation, which enables studying the memory hierarchy with a faithful representation of a modern x86 multicore processor. The simulated hardware is capable of running unmodified operating systems and user software, which generates authentic memory access patterns for memory hierarchy studies. To demonstrate the capabilities of our infrastructure we include a series of experimental examples that utilize the cache, DRAM main memory, and nonvolatile memory modules.

Introduction

The rise of multicore systems has shifted the primary bottleneck of system performance from the processor to the memory hierarchy, accelerating the gap that had already existed between processor and memory performance (the memory wall). Previously, the memory wall problem was the result of the increasing frequencies of CPUs relative to the latency of the memory system, which meant that CPUs were losing more processing time waiting on memory accesses. However, as processor frequency improvements stalled and with the introduction of multicore systems, a more urgent problem was created since the current memory system cannot scale at the same rate as the number of cores. Therefore, in modern systems there is actually much less bandwidth and capacity per core than there was a few years ago. This trend can be seen in Figure 1. This problem, combined with the existing operating frequency problem, has led to the memory hierarchy becoming the dominant source of slowdown in the system. To address the increased need for capacity, systems are now relying more on solid state drives and other high performance storage systems, exacerbating the latency problem of the memory system due to the increased frequency of references to the slower storage system. Finally, since multicore systems are running threads in different address spaces with different access patterns, there is less locality of reference for the cache hierarchy to exploit. This implies that overcoming the multicore memory wall problem requires examining the entire memory hierarchy from the cache system down to the storage system.

"…overcoming the multicore memory wall problem requires examining the entire memory hierarchy…"

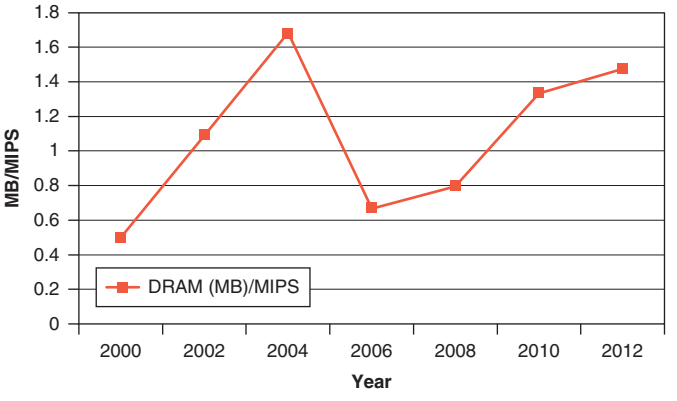

Figure 1: DRAM capacity (MB)/processor speed(MIPS) per core for a typical system (Source: University of Maryland, 2013)

In addition to the strain on memory system capacity and bandwidth that has been introduced by multicore chips, memory system capacity is also limited by scaling problems at the device level. For DRAM, as the memory cells shrink, the charge that can be stored on the capacitor becomes very small and the pass transistor leakage increases, which reduces the retention time of the cell and requires more complex peripheral circuitry to detect the smaller charge. For flash memory, as the dielectric of the floating gate shrinks, the amount of damage during program-erase cycles that can be tolerated decreases and the cells wear out faster.[1] Additionally, since control circuitry has analog components that are difficult to scale down, as the DRAM and flash cell size decreases, the control circuitry takes up a larger percentage of the chip area relative to the memory array. Architects have attempted to address device scaling problems by adding more devices with technologies like FB-DIMM and Buffer on Board, as well as technologies in currently development like the Hybrid Memory Cube.[2] However, these solutions require additional hardware to be designed and added to the memory system, making them currently prohibitively expensive for most applications. New memory technologies have also been suggested that might eventually provide a solution to the capacity problem but these technologies are not yet competitive with existing technologies in terms of cost or capacity.[17] Meanwhile, software is not helping to alleviate the situation, because application working sets continue to increase in size. In recent years, big data applications such as bioinformatics and graph analytics have only accelerated the increasing demand for faster and more scalable storage systems. This has also contributed to the rapid adoption of solid state drives. However, much of the storage system's software and hardware infrastructure was constructed around assumptions of millisecond access latencies and, as a result, fails to efficiently utilize the new high performance storage solutions being implemented. In order to meet the new challenges posed by big data applications, the storage system needs to be reworked from the OS file system down to the hardware interfaces. Finally, as the

"…big data applications such as bioinformatics and graph analytics have only accelerated the increasing demand for faster and more scalable storage systems."

community pushes towards exascale computers, the power and resilience limitations of the current memory system are becoming more pronounced. [3] If an exascale-sized main memory system were constructed using today's technology, then just that component alone would consume the entire system power budget. Furthermore, given the current probability of failure in memory system components, as the number of components approach the numbers needed for exascale, the probability of a failure somewhere in the system approaches 1. This means that if an exascale computer were built with today's memory technology, not only would it use too much energy, it would also be breaking constantly. Therefore, to enable the push to exascale it is imperative that new, more energy-efficient and resilient memory technologies and architectures be developed.

"…the feedback between the various components of the system is vital to understanding performance."

In order to overcome these problems, new architectures and software need to be developed and evaluated. Since the new solutions will involve multiple aspects of the system, the feedback between the various components of the system is vital to understanding performance. For example, many researchers are studying how to integrate nonvolatile memory into the system as a first class citizen, which involves both the hardware and the software. Trace-based simulation has been used in the past to study these kinds of architecture problems. Unfortunately, trace-based simulation does not capture the feedback loops between software and hardware. One way to produce these feedback loops is to build a real-world prototype. However, due to the engineering effort required, real-world prototypes are impractical and costly for studying large design spaces. Full system simulation models those complex interactions and can provide valuable insights into the dynamic behavior of a variety of system designs. Previously, no full system simulator existed that could study all levels of the memory and storage hierarchy. In this article, we describe our simulation infrastructure that addresses this need by providing a full system simulator capable of modeling the entire processor and memory hierarchy, including the storage system.

Simulator Description

Our memory hierarchy simulation infrastructure is an extension of the MARSSx86 full system simulation environment[4] developed at SUNY Binghamton. We utilize MARSS to simulate the microprocessor and other non-memory hierarchy components of the system. The memory infrastructure builds on top of the prior MARSS memory hierarchy and incorporates detailed simulations of every level of the hierarchy including the cache, the main memory system, and the storage system. The cache simulator is an extended version of the existing cache simulation in MARSS that allows for heterogeneous technologies at different levels of the cache hierarchy. For traditional DRAM-based main memory systems, our simulation environment uses DRAMSim2, which is a detailed, cycle-accurate DRAM memory system simulator developed by our lab[13]. For nontraditional hybrid nonvolatile/DRAM memory systems our simulation environment uses two modules, HybridSim and NVDIMM, which simulate the memory controller and

nonvolatile DIMMs that would be used by such a system. The hybrid memory components can also be reconfigured to simulate solid state drives. Figure 2 shows the overall structure of our simulation environment, including its constituent modules and how they communicate with one another.

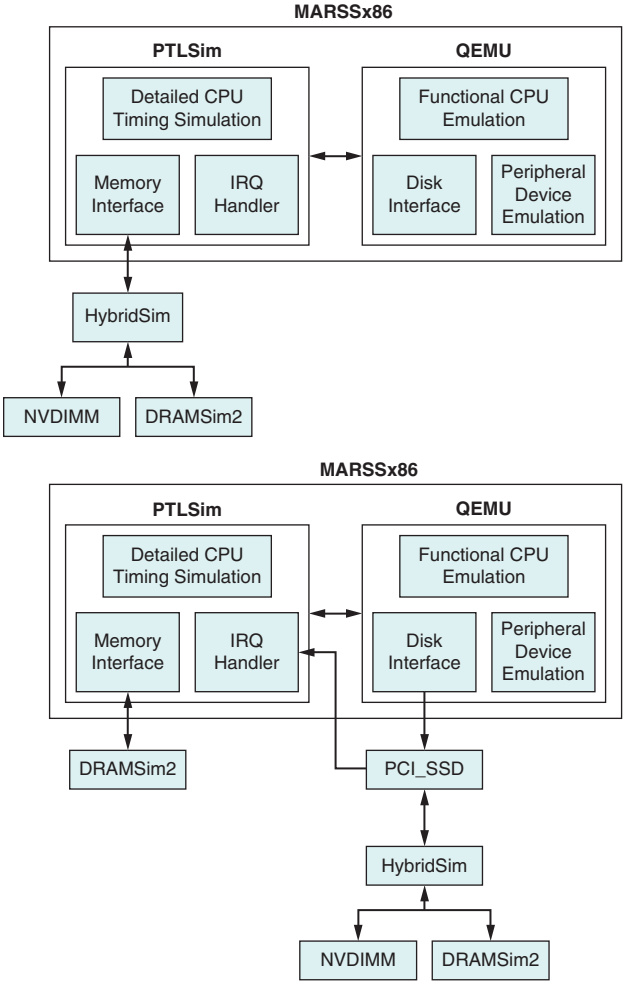

Figure 2: Block diagram of simulation environment for hybrid memory (top) and SSD (bottom) (Source: University of Maryland, 2013)

MARSS

MARSS is designed to simulate a modern x86 computer system. MARSS utilizes PTLSim to simulate the internal details of the processor. PTLSim is capable of simulating a multicore processor with the full details of the pipeline, micro-op front end, reorder buffers, trace cache, and branch predictor. In addition, PTLSim also simulates a full cache hierarchy and can implement several cache coherency protocols. For the hardware that is not explicitly simulated, such as disks or the network card, MARSS uses the QEMU emulation environment. MARSS is able to boot full, unmodified operating systems, such as any Linux distribution, and then run unmodified benchmarks. We selected MARSS as the basis for our memory hierarchy simulation infrastructure because of its ability to simulate

"We selected MARSS as the basis for our memory hierarchy simulation infrastructure because of its ability to simulate both the user programs and the operating system functionality…"

both the user programs and the operating system functionality, while most other simulation environments are only capable of simulating user-level instructions. Therefore, in addition to being the most realistic simulation environment possible, MARSS can be used to study the behavior of the operating system, which we view as vital to solving the problems of future memory and storage systems.

Cache Simulation

While PTLSim already provides an SRAM-based cache simulation, studying other technologies is vital because of the power, bandwidth, and capacity problems that arise in the design of the memory hierarchy for future systems. Memory technologies such as SRAM, STT-RAM, and eDRAM have been considered for implementing on-die LLCs. Though they all have low read latency and high write endurance, they can be very different for other performance characteristics. For instance, SRAM is low density and has high leakage current, STT-RAM has high write latency and write energy consumption, and eDRAM requires refresh. Additionally, due to the very different inherent characteristics of each of the memory technologies, researchers have proposed various power and performance optimization techniques. Therefore, in order to make useful comparisons between SRAM, STT-RAM, and eDRAM LLCs, we expand MARSS with the following:

1. We integrate a refresh controller into MARSS to support eDRAM LLCs.
2. In addition to the parameterized cache access time, we expand MARSS with parameterized cache cycle time, tag access latency, and refresh period. Separating cycle time and tag access latency allows the user to evaluate pipelined caches and sequentially accessed caches (such as when data array access is skipped on a tag mismatch). We also modify MARSS to support asymmetric cache read and write latencies. This property is required to evaluate STT-RAM caches realistically.
3. We integrate dead line predictors to enable low power modes for SRAM and eDRAM caches.

"These changes allow our environment to investigate future cache designs incorporating new technologies and techniques."

These changes allow our environment to investigate future cache designs incorporating new technologies and techniques.

DRAM Main Memory Simulation

Since the DRAM-based main memory system has a large number of configuration and timing parameters, such as the command and data queues, address mappings, refresh timings, low power modes, activate and pre-charge periods, and so on, choice of one or another scheme could have drastically different power or performance implications.[14] Therefore, DRAMSim2, a cycle-accurate JEDEC DDRx memory system simulator, was developed.[12][13] It models the memory controller, memory channels, DRAM ranks, and banks. The DRAMSim2 timing behavior has been compared and validated against Verilog-based device models published by DRAM vendors.

Recently, JEDEC published the next generation DDR4 standard.[15] DDR4 devices could operate at double the speed of previous generation DDR3 chips,

and moreover DDR4 will have additional features enabling low power and high memory capacity. DDR4 devices will have banks separated into multiple bank-groups to facilitate higher bandwidths and greater bank-level parallelism. However, since banks within a bank-group share some peripheral circuitry, requests to banks of the same bank-group takes longer time than banks on different bank-groups. We modified the DRAMSim2 memory controller to incorporate these DDR4 specific changes.

Power dissipated due to DRAM represents a substantial portion of the total system power budget, as the main memory capacity and bandwidth increases to satisfy requirements of the current and future data-intensive applications. Therefore, to study the tradeoffs involved with switching to various DRAM low power modes, such as active, power-down, self-refresh, and deep power-down , requires accurate switching time as well as the current drawn during each mode. Furthermore, refresh command scheduling could also potentially affect the switching to low power modes. We have augmented DRAMSim2 with detailed low power modes and a range of refresh policies, allowing users to study the performance and power tradeoff when using different low power modes and refresh methods.

"We have augmented DRAMSim2 with detailed low power modes and a range of refresh policies,"

Nonvolatile Memory Simulation

Recently many designs have been proposed that utilize nonvolatile device based DIMMs to address the capacity issues of the main memory system. For DIMMs that are not made using DRAM parts, we use NVDIMM, which is capable of simulating DIMMs made from a wide variety of technologies. This is possible because most nonvolatile technologies share many common features and differ in only a few parameters. For instance, both flash and Phase Change Memory (PCM) feature asymmetric reads and writes. To allow for these differences, NVDIMM has a wide variety of options that can be used to shape the behavior of the system. Some technology-specific options include access latencies, device interface widths, address mapping policies, and wear leveling policies. For example, in flash a dynamic mapping scheme is used so that dirty pages can be set aside to be erased during idle cycles by a garbage collection process, enabling faster modifications of existing data. This scheme was chosen because the erase time for flash is prohibitively long even for basic storage applications. Early architectures for PCM, on the other hand, have been designed with a simpler static mapping scheme that does not require a garbage collection process because its erase is considerably faster than flash's.

In addition, other options have been included in NVDIMM to enable investigations into the effects of organization, scheduling, and timing. A good example of such a study is to determine how many devices of a given type can be included on a DIMM before the host interface channel (such as DDR3 or SATA) is saturated. By enabling both device and architecture level investigations, NVDIMM allows our memory hierarchy simulation infrastructure to study different methods for integrating nonvolatile memory into a computer system.

"…NVDIMM allows our memory hierarchy simulation infrastructure to study different methods for integrating nonvolatile memory into a computer system."

Nonvolatile Memory Integration

There are two primary ways to integrate nonvolatile memory into a computer system below the cache level, as illustrated in Figure 3. The first method is the traditional storage route, which uses the same software and hardware abstractions and protocols as hard disk drives. The second method is to tie the nonvolatile memory directly into the memory controller. Our memory hierarchy simulation infrastructure is designed in such a way that you can utilize a common set of modules to simulate both integration methods, which enables the ability to make fair comparisons between the two.

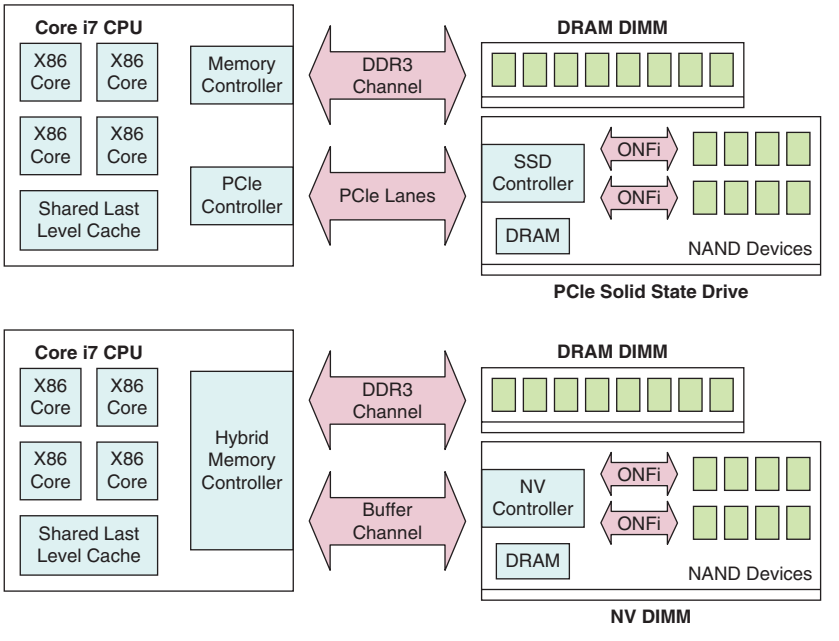

Figure 3: System design for SSD (top) and hybrid memory (bottom) (Source: University of Maryland, 2013)

In both disk-like and memory-like integration methods, since the nonvolatile memory typically has long latencies, a faster memory such as DRAM or SRAM is utilized as a buffer or cache. We provide the HybridSim module to simulate this aspect of the system. HybridSim uses NVDIMM as its backing store and DRAMSim2 as its cache. HybridSim's features enable the study of a variety of cache replacement policies, prefetching policies, and hardware/software co-design (for example, having the memory controller and operating system work together to manage nonvolatile memory).

"HybridSim's features enable the study of a variety of cache replacement policies, prefetching policies, and hardware/software co-design…"

When HybridSim is simulating a memory-like integration method for nonvolatile memory, also known as a hybrid main memory, it interacts with the memory controller of the base MARSS system to capture addresses and bypass its simpler memory model. HybridSim then performs its caching functions and sends requests to DRAMSim2 or NVDIMM to implement requests. When the requests complete, HybridSim sends callbacks to the MARSS memory controller to indicate that a request is done and allow the processor to make progress at the appropriate clock cycle.

When HybridSim is simulating a disk-like integration method, it receives disk requests from MARSS and then later raises an I/O interrupt to indicate a request is complete. This works exactly like a modern solid state drive. We also provide an additional module called PCI_SSD to simulate the host interface for a modern SATA or PCIe SSD and to allow the user to configure various options including the number of lanes, half or full duplexing, an optional two-level interface (such as Intel Direct Media Interface to SATA), frequency, and protocol overhead. Our SSD simulation also ties in with our DRAMSim2 main memory simulation to perform direct memory access operations to DRAM before or after a disk request occurs. This process of disk simulation is also compatible with simulators for conventional hard disk drives like DiskSim[5] and HDD simulation could be achieved by simply modifying the PCI_SSD module.

"Our SSD simulation also ties in with our DRAMSim2 main memory simulation to perform direct memory access operations…"

Simulation Variability and Warm-Up

Full system simulation introduces some additional sources of complexity and nondeterminism that can lead to inaccurate results if they are not dealt with properly. In particular, just as in a real system, the OS introduces nondeterminism into the simulation as a result of timing variation (for example, interrupt arrival time) from run to run. This problem can be reduced by utilizing checkpoints of the system state, which MARSS enables using the QEMU snapshot mechanism. Another source of complexity is how to properly warm up the caches and other state (such as NVDIMM's address mapping) for novel memory hierarchy architectures. We provide a generic mechanism for warm-up utilizing state files that can be saved during a warm-up period or generated by scripts and then restored at the beginning of the region of interest. An example of this warm-up process can be seen in Figure 4.

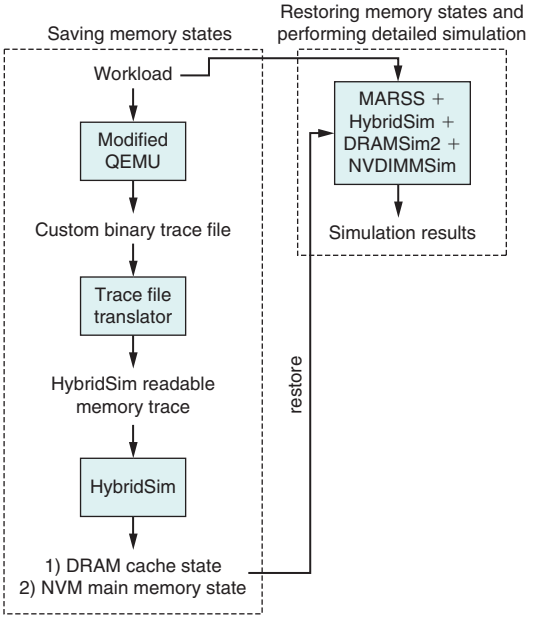

Figure 4: An example of a complete warm-up (Source: University of Maryland, 2013)

Baseline Configuration

The baseline configuration for the following experiments is a quad-core, out-of-order system, with cache organization similar to the Intel® Core™ i7. The cache experiments below use this processor with a modified LLC to incorporate new memory technologies. The cache experiments also utilized the baseline DRAM main memory configuration. These baseline configurations are shown in Table 1.

Processor	4-core, issue width = 4, 2 GHz
L1I (private)	128 KB, 8-way, 64-B block size
L1D (private)	128 KB, 8-way, 64-B block size
L2 (private)	2 MB, 8-way, 64-B block size
L3 (shared) (if present)	8 MB, 16-way, 64-B block size
DRAM (if used as cache)	512 MB, 64-way, 4-KB page size
DRAM (if used as main memory)	1 GB, DDR3-1333
Nonvolatile main memory	8 GB, 4-KB page size, PCIe 3.0 16 Lane equivalent bandwidth

Table 1: Baseline Configuration
(Source: University of Maryland, 2013)

The DRAM examples also utilize the baseline processor and cache shown in Table 1.

For the hybrid and SSD experiments, an 8-GB NVM is considered, with a 512-MB DRAM cache in front of it. The nonvolatile DIMM organization has 1 channel, 64 dies per channel, 2 planes per die, 4,096 blocks per plane, 64 pages per block, and each page is 4 KB. All transfers between the NVM and the DRAM occur at the page granularity. The timing parameters for the nonvolatile memory are based on MLC flash numbers.[6] The DRAM cache, also in the form of a DIMM, is organized as 1 channel, 1 rank per channel, 8 banks per rank, 8,192 rows per bank, and 1,024 columns per row. All transfers between the DRAM and the L2 cache occur at the L2 cache line granularity (64 B). DRAM timing parameters are based on a Micron datasheet.[7] All devices are 8 bits wide.

For these experiments we use the GUPS benchmark and a random access micro-benchmark called mmap developed by our lab as well as selected benchmarks from the NAS benchmark suite, the SPEC benchmark suite, and the PARSEC benchmark suite.[8][9][10][11] These benchmarks were selected because they have a large working set size and are memory intensive.

Experiments

The following experiments demonstrate examples of the wide variety of studies that can be performed using the various modules of our environment. For the processor cache, we present energy and execution time data for last-level caches constructed using different memory technologies for a several benchmarks. To demonstrate the capabilities of the DRAM system portion of the simulator, we have included power and instructions-per-cycle data for similar sets of several benchmarks. Finally, we exhibit the features of the nonvolatile memory portions of our environment with data showing the effects of additional bandwidth, prefetching, working set size, and memory system traffic volume on system performance. Table 1 contains the baseline configuration details that are common to all of the experiments.

"…we compare the LLC energy consumption and system performance when using SRAM, STT-RAM, and eDRAM."

Caches

As a case study, we compare the LLC energy consumption and system performance when using SRAM, STT-RAM, and eDRAM. The LLC is a 32-nm, 32-MB, 16-way write-back cache that is partitioned into 16 banks and uses 64-byte blocks. It is also pipelined and sequentially accessed.

Figure 5 illustrates the normalized energy breakdown of LLCs based on SRAM, STT-RAM, and eDRAM. We include the results for "regular" implementations (without power-optimization) and "low power" implementations. For instance, "regular" SRAM uses high performance transistors to implement the entire cache without power gating; "regular" STT-RAM uses storage-class STT-RAM technology, which has a long retention time but requires high write energy; and "regular" eDRAM uses the conventional periodic refresh method. On

the other hand, low power SRAM uses dead line prediction[18], power gating, and low leakage CMOS for the memory cells[19] to reduce leakage power; low power STT-RAM uses device optimization techniques to reduce write energy by sacrificing data-retention time[17]; and low power eDRAM uses dead line prediction to reduce the number of refresh operations. The impact of different memory technologies and implementations on system performance is shown in Figure 6.

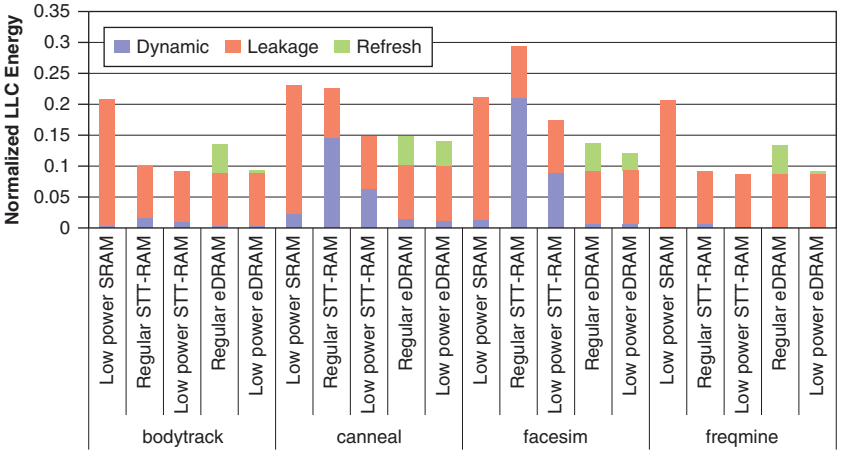

Figure 5: Normalized LLC energy breakdown with respect to various memory technologies. The results are normalized to regular SRAM (not shown). Note that regular SRAM dissipates 5x more power on average
(Source: University of Maryland, 2013)

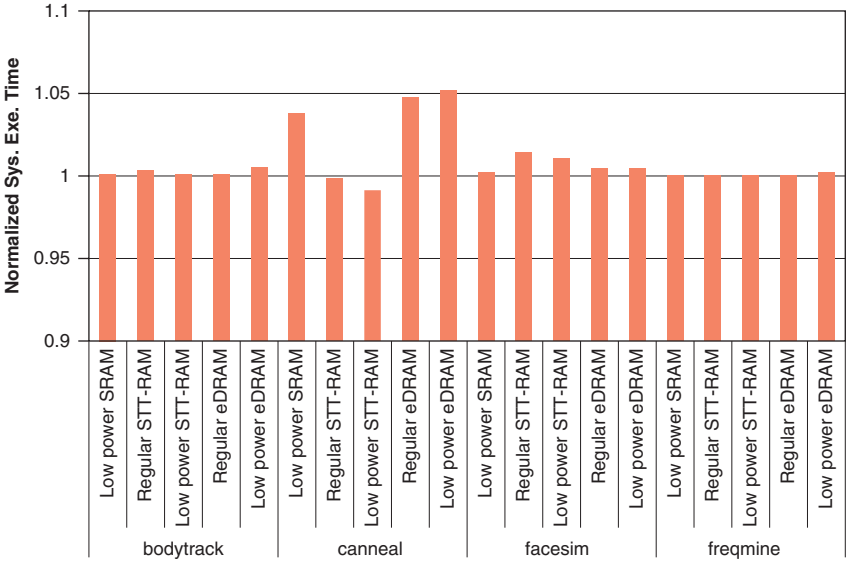

Figure 6: Normalized system execution time with respect to various memory technologies. The results are normalized to regular SRAM (not shown)
(Source: University of Maryland, 2013)

DRAM

As an interesting case study of a DRAM-based main memory system, we show the impact of refresh when device size is increased from small 1-Gb to future big 32-Gb chips. We simulated few SPEC2006 benchmarks in region of interest (RoI) for 1 billion instructions, assuming both with and without refresh enabled. Figure 7 presents the energy contribution separated for each type of operation, that is: read and write, activate and pre-charge, background and refresh operations. The Y-axis representing energy is normalized to the corresponding 1-Gb device values for each benchmark. The background and refresh energy portion increases for higher density devices, because of the greater number of peripheral circuitry and cells to be refreshed as device size increases. Since with DRAM density, the number rows also increases, this leads to more frequent refresh commands to be scheduled, and therefore leads to a degradation of the memory performance and latency. Figure 8 shows the percentage degradation of system performance (IPC) and the average latency increase due to refresh operations as the size of DRAM devices vary.

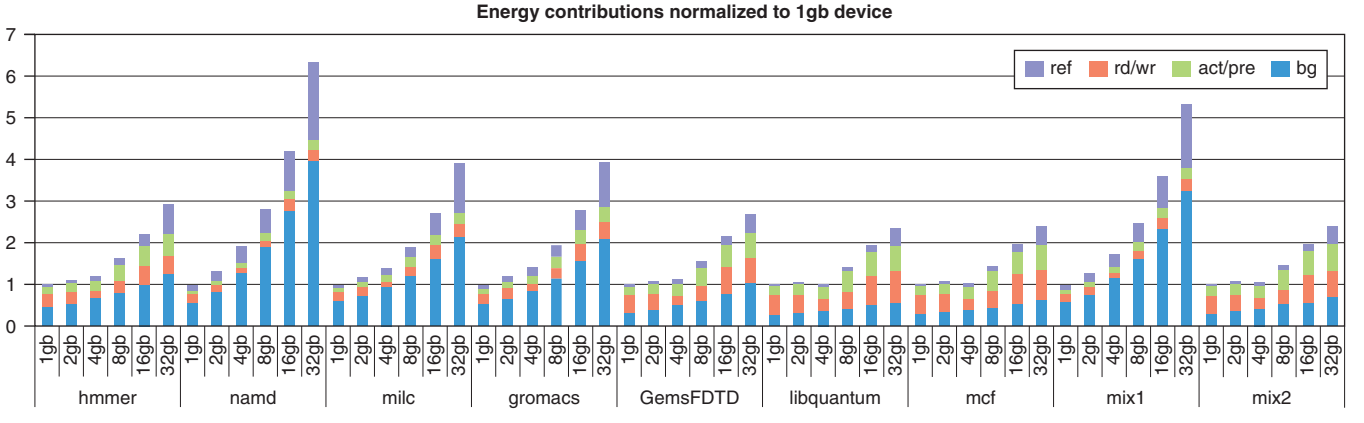

Figure 7: Energy contributions separated for each operation type normalized to the 1-Gb DRAM device size. Refresh and background energy consumption increases when DRAM density gets higher
(Source: University of Maryland, 2013)

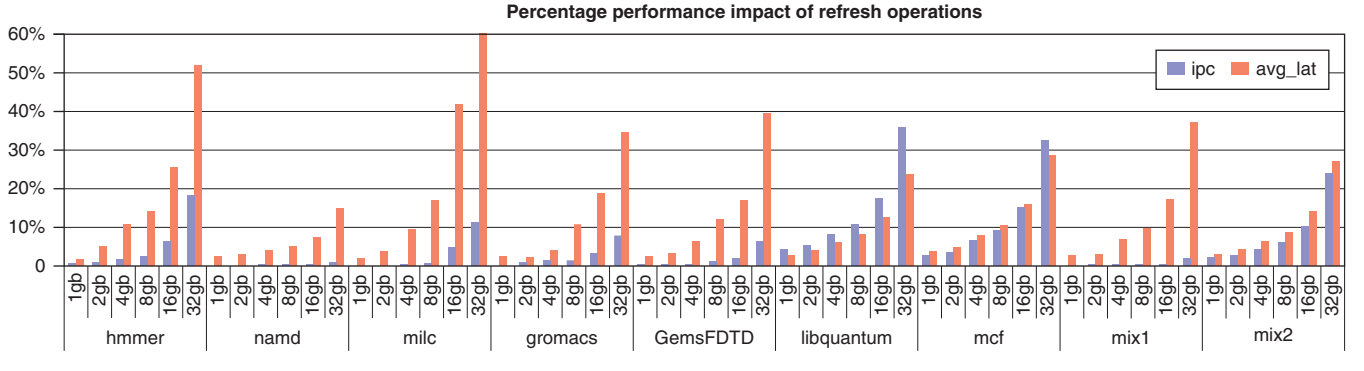

Figure 8: Percentage of refresh penalty measured using Instructions Per Cycle (IPC) of the entire system and average latency of the memory system. For higher density devices, the performance penalty increases sharply
(Source: University of Maryland, 2013)

Nonvolatile Memory

An important type of study for future memory systems is to understand how the system reacts to changing the working set size and volume of accesses. This is especially important in hybrid main memory systems because nonvolatile memory latencies can be significantly slower than traditional SRAM and DRAM. The Giga-Updates Per Second (GUPS) implementation from Sandia National Laboratories is an ideal benchmark to study such access patterns since it takes the working set size and number of accesses as parameters, unlike many other benchmarks that assume a constant pattern for memory accesses. GUPS creates a large table and then performs a series of updates on pseudorandom locations within that table. In this experiment we chose table sizes of 256 MB, 512 MB and 1 GB. The DRAM cache in our test system was 512 MB. Our choice of table sizes allows us to see the effect on system performance when the table fits in the DRAM cache easily, when the table is approximately the same size as the DRAM cache to cause some swapping between the DRAM cache and the nonvolatile backing store, and when the table is two times the size of the DRAM cache to cause a significant number of DRAM cache misses. We also vary the number of updates from 1000 to 5000 in increments of 1000 to show the effect of different volumes of memory traffic on system performance. Finally, we included data for systems that incorporate the nonvolatile memory as both a hybrid memory and as a traditional SSD. From the results in Figure 9, we can see that for the SSD configuration as the table grows larger than the 512 MB DRAM and more accesses must go to the slower flash swap space, system performance suffers as would be expected. However, for the hybrid memory version, performance is not dependent on the table size. This is because Linux sees the 8 GB backing store as the main memory address space and allocates the entire table inside this space. Initially, this table is not present in the DRAM cache because it has been accessed yet. When the table size is twice the size of the DRAM, the performance of the Hybrid implementation becomes much better than the SSD implementation. This is because the SSD has more overhead for its accesses to the swap space than the Hybrid has for its accesses to the flash.

"…the SSD has more overhead for its accesses to the swap space than the Hybrid has for its accesses to the flash."

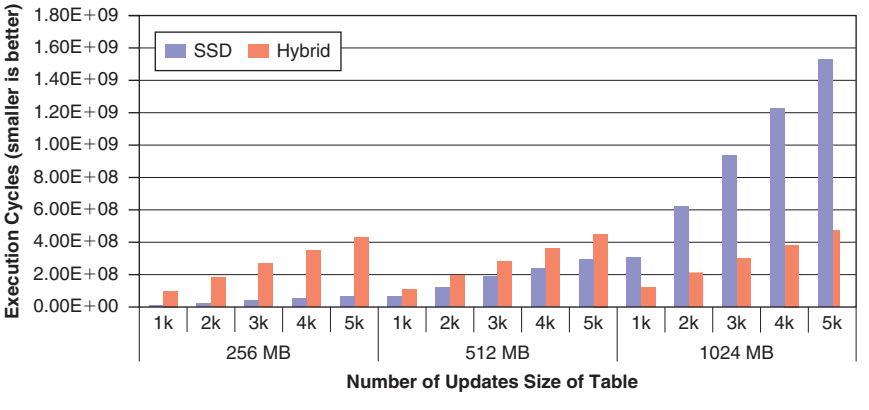

Figure 9: Execution time of GUPS when table size and number of updates are varied (smaller is better)
(Source: University of Maryland, 2013)

"Optimizing the performance of the nonvolatile backing store is another important area of study for future memory systems."

Optimizing the performance of the nonvolatile backing store is another important area of study for future memory systems. One area of potential performance gain is the interface of the nonvolatile devices used to create the backing store. To show the effect of improving the bandwidth provided by these interfaces, we utilize an in-house micro-benchmark called MMAP. MMAP works by first defining a large memory mapped file that is opened with the mmap() system call in Linux and then it accesses this file randomly. This benchmark is well suited to bandwidth studies because it is single threaded and therefore provides a clear picture of the effect of a minor change without much noise from other system threads. Additionally, since MMAP is designed to force misses to the DRAM cache as often as possible, which causes only one 64-byte access within each 4-KB page, it maximally stresses the host interface and device channels in the backing store. This is a worst-case scenario for the memory system because it generates a large volume of random accesses that are not fully utilized by the cache. This is the reason for the low observed IPC. For this experiment, we vary the clock rate of the interface of a device (the amount of time it takes to transmit 8 bits of data) from 0.05 ns to 10 ns. In addition, we also utilize a basic sequential prefetching algorithm to generate more accesses and place greater pressure on the devices. We vary the number of additional pages that are prefetched by our algorithm from 4 to 8 to 16. As was the case in the previous example, we also include data for both a hybrid-style integration of the nonvolatile memory and an SSD-style integration. In Figure 10, we can see that both faster device interfaces and larger prefetching windows help to improve the system performance. We do not use the prefetching in HybridSim for the SSD version of the experiment because prefetching is performed by the operating system for disk accesses. It is also important to note that there is less nondeterminism in these results than in the previous example because this example is single threaded, which eliminates nondeterminism introduced by the OS scheduler when it has to schedule multiple threads. There is still some minor nondeterminism in this experiment's results, but that is what one would expect from a real system.

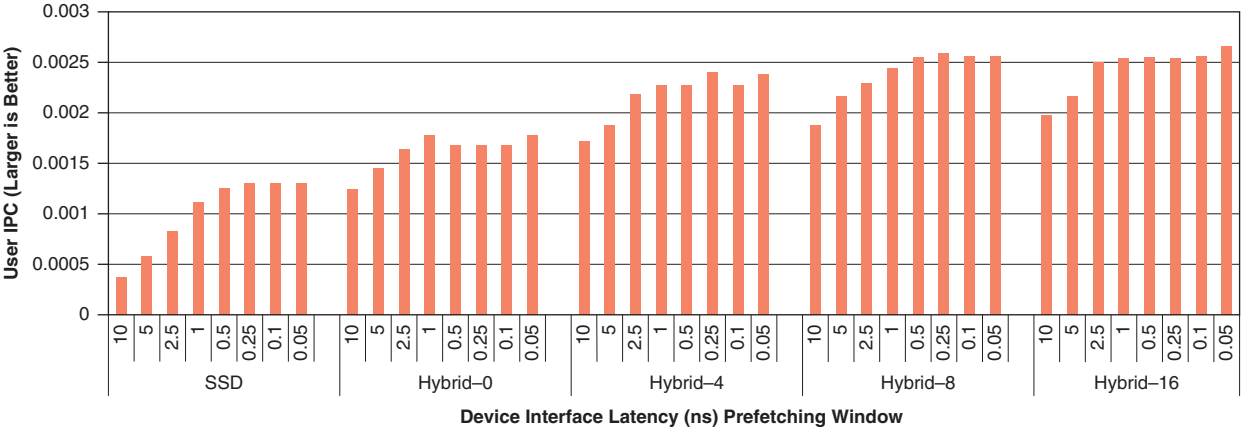

Figure 10: Performance of MMAP with varying bandwidth and prefetching window size (Source: University of Maryland, 2013)

Conclusion

In this work, we have introduced a complete memory hierarchy simulation environment that is capable of accurately simulating the processor cache, the DRAM main memory system, and nonvolatile memory, whether it is implemented as a hybrid memory or as an SSD. We have shown the utility of this infrastructure for solving future memory hierarchy design problems by presenting example experiments that demonstrated multiple last-level cache cell technologies, DRAM refresh schemes, and nonvolatile memory integration methods.

References

[1] Laura M. Grupp, John D. Davis, and Steven Swanson. 2012. "The bleak future of NAND flash memory." In Proceedings of the 10th USENIX conference on File and Storage Technologies (FAST'12). USENIX Association, Berkeley, CA, USA, 2–2.

[2] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, "Buffer On Board memory systems," in Proceedings of the 39th Annual International Symposium on Computer Architecture, ser. ISCA '12, 2012.

[3] US Department of Energy Office of Science. "The Opportunities and Challenges of Exascale Computing." Summary Report of the Advanced Scientific Computing Advisory Committee (ASCAC) Subcommittee. Fall 2010.

[4] A. Patel et al., "MARSSx86: A Full System Simulator for x86 CPUs," in Design Automation Conference 2011 (DAC'11), 2011.

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

[6] Micron Technology. "128Gb SLC Flash Datasheet." 2012. [Online]. Available: http://www.micron.com/parts/nand-flash/mass-storage/mt29f128g08akcabh2-10

[7] Micron Technology. "4Gb DDR3 SDRAM Datasheet." 2009. [Online]. Available: http://www.micron.com/parts/dram/ddr3-sdram/mt41j256m16re-125

[8] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. 2008. "The PARSEC benchmark suite: characterization and architectural implications." In Proceedings of the 17th international conference on Parallel architectures and compilation techniques (PACT '08). ACM, New York, NY, USA, 72–81.

[9] NASA Advanced Supercomputing Division. "NAS Parallel Benchmarks." 2012. [Online]. Available: http://www.nas.nasa.gov/publications/npb.html

[10] Standard Performance Evaluation Corporation. "SPEC2006 CPU Benchmarks." 2012. [Online]. Available: http://www.spec.org/cpu2006/

[11] Sandia National Labs. "RandomAccess GUPS (Giga Updates Per Second)." 2012. [Online]. Available: http://www.sandia.gov/~sjplimp/algorithms.html

[12] David Wang, Brinda Ganesh, Nuengwong Tuaycharoen, Katie Baynes, Aamer Jaleel, and Bruce Jacob. "DRAMsim: A memory-system simulator." SIGARCH Computer Architecture News, Vol. 33, No. 4, pp. 100–107. September 2005.

[13] P. Rosenfeld, E. Cooper-Balis, and B. Jacob, "DRAMSim2: A Cycle Accurate Memory System Simulator," Computer Architecture Letters, Vol. 10, No. 1, pp. 16–19, Jan.-June 2011.

[14] S. Srinivasan, L. Zhao, B. Ganesh, B. Jacob, M. Espig, and R. Iyer. "CMP memory modeling: How much does accuracy matter?" Proc. Fifth Annual Workshop on Modeling, Benchmarking and Simulation (MoBS), pp. 24–33. Austin TX, June 2009.

[15] JEDEC, "DDR4 SDRAM Standard," 2012. [Online]. Available: http://www.jedec.org/sites/default/files/docs/JESD79-4.pdf

[16] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in Proceedings of the 36th Annual International Symposium on Computer Architecture, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 24–33.

[17] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan. "Relaxing non-volatility for fast and energy-efficient STT-RAM caches," High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on, Vol., no., pp. 50–61, 12–16 Feb. 2011.

[18] Stefanos Kaxiras, Zhigang Hu, and Margaret Martonosi. "Cache decay: exploiting generational behavior to reduce cache leakage power." In Proceedings of the 28th annual international symposium on Computer architecture (ISCA '01). ACM, New York, NY, USA, 240–251.

[19] K. Roy, S. Mukhopadhyay, and H. Mahmoodi-Meimand, "Leakage current mechanisms and leakage reduction techniques in deep-submicrometer CMOS circuits," Proceedings of the IEEE, Vol. 91, No. 2, pp. 305–327, Feb. 2003.

Author Biographies

Jim Stevens received a BS degree in computer engineering from the University of Kansas in 2006, an MS degree in computer science from the University of Arkansas, Fayetteville, in 2009, and is currently pursuing a PhD in computer science at the University of Maryland, College Park. His research interests include memory controller design and adapting operating systems for nonvolatile memories.

Paul Tschirhart received his BS degree in computer engineering from the University of Virginia in 2007. He is currently pursuing a PhD in computer and electrical engineering at the University of Maryland, College Park. His research interests include memory controller design, memory system architecture, and SSD design.

Mu-Tien Chang received the BS and the MS in electronics engineering from National Chiao Tung University, Hsinchu, Taiwan, in 2006 and 2008, respectively. He is currently pursuing a PhD in electrical and computer engineering at the University of Maryland, College Park. His research interests include memory circuit and processor cache design.

Ishwar Bhati received B.Tech. in electronics and communication engineering from Indian Institute of Technology, Guwahati, India, in 2005. He worked in the VLSI/ASIC industry as design and verification engineer for five years. He is currently pursuing a PhD in electrical and computer engineering at the University of Maryland, College Park. His research interests include energy-efficient memory systems and high performance computing.

Peter Enns is currently pursuing a PhD in linguistics at the University of Maryland with a concentration in computational linguistics and natural language processing. He received a BS in computer engineering from the University of Maryland in 2011 with honors (summa cum laude). While he was an undergrad, Peter studied nonvolatile memory systems with Dr. Bruce Jacob in Maryland's Memory Systems Research Lab.

James Greensky is currently a software engineer in the Memory Architecture Lab (MAL) within Intel Labs. James received his BS and MS degrees in computer science and is currently pursuing a PhD in the area of computer architecture from the University of Minnesota.

Zeshan Chishti received the BSc (Hons) degree in electrical engineering from the University of Engineering and Technology, Lahore, Pakistan, in 2001, and a PhD in computer engineering from Purdue University in 2007. He is a Research Scientist at Intel Labs, Hillsboro, Oregon. His research interests include microarchitecture, energy-efficient memory systems, and cache hierarchies for chip multiprocessors.

Shih-Lien Lu received his BS in EECS from UC Berkeley, and MS and PhD both in CSE from UCLA. He is a principal researcher and leads the memory architecture team at Intel Labs. From 1984 to 1991 he was on the MOSIS project at USC/ISI, which provides research and education community VLSI fabrication services. He was on the faculty of the ECE Department at the Oregon State University from 1991 to 2001. His research interests include computer microarchitecture, memory circuits, and VLSI systems design.

Bruce Jacob received the AB degree in mathematics from Harvard University in 1988 and the MS and PhD degrees in CSE from the University of Michigan in Ann Arbor in 1995 and 1997, respectively. He also worked for two successful startup companies: Boston Technology and Priority Call Management; at Priority Call Management he was the initial system architect and chief engineer. He is a professor of electrical and computer engineering at the University of Maryland in College Park, and he is currently visiting at the University of Siena, Italy, where he is working on memory issues for many-core systems. He is a recipient of a US National Science Foundation CAREER award for his work on DRAM, and he is the lead author of an absurdly large tome on the topic of memory systems. His research interests include memory systems, operating systems, and designing electric guitars.