Theme Article

Performance Assessment of Emerging Memories Through FPGA Emulation

Abhishek Kumar Jain, Scott Lloyd, and Maya Gokhale
Center for Applied Scientific Computing, Lawrence Livermore National Laboratory

***Abstract*—Emerging memory technologies offer the prospect of large capacity, high bandwidth, and a range of access latencies ranging from DRAM-like to SSD-like. In this paper, we evaluate the performance of parallel applications on CPUs whose main memories sweep a wide range of latencies within a bandwidth cap. We use an FPGA emulator, the logic in memory emulator (LiME) to accelerate evaluation. The LiME framework uses a multiprocessor system on chip (MPSoC) combining multicore CPU, integrated memory controller, and FPGA fabric, and enables emulation orders of magnitude faster than software simulation. Our paper highlights the performance impact of higher latency on concurrent applications and identifies conditions under which future high latency memories can effectively be used as main memory.**

After many years of R&D, innovations in memory technology are appearing in commercial products. At one end of the capacity spectrum, stacked DRAM such as high bandwidth memory and hybrid memory cube offer a significant jump in bandwidth but have the small capacity (8–32 GB) and slightly longer latencies than high performance DDR. On the other end, high capacity storage class memories (SCMs) are emerging with unique characteristics such as asymmetric read/write latency, wear from writes, and potentially high error rates. Examples of these persistent memories include PCM, 3-D XPoint,[1] RRAM, and MRAM. Combinations of these memories are being proposed for future computing systems as an alternative to high performance DDR.[2]

In this paper, we focus on the memory latency spectrum to study application performance when main memory latency is varied by a factor of nearly 18-fold, from 45 to 800 ns. Using a multiprocessor system on chip (MPSoC) combining a multicore CPU with FPGA logic in the logic in memory emulator (LiME)[3] framework, we have emulated the performance of compute servers running data intensive benchmarks and

Digital Object Identifier 10.1109/MM.2018.2877291
Date of publication 8 November 2018; date of current version 21 February 2019.

Published by the IEEE Computer Society

mini-applications. We have conducted parameter sweeps in which the main memory latencies model the entire range from fast DRAM to SCM. LiME enables emulation over multiple latencies and multiple levels of concurrency many orders of magnitude faster than software simulation, at only a 20× slowdown over a 2.75-GHz multicore CPU. LiME enables rapid evaluation of a large parameter space and can run applications both standalone and under a full Linux stack. Another approach is presented by the Quartz emulator[4] to emulate a wide range of nonvolatile memory latencies and bandwidth characteristics on certain Xeon CPUs by modifying DRAM timing settings in the BIOS. This method shows emulation error of 0.2%–9%. Our platform uses ARM processors and emulates latencies at 0.16 ns increments. Additionally, our platform can write memory event traces to a separate memory without affecting the main memory where the workload is run. O'Conner *et al.*[5] propose innovations in DRAM to better meet bandwidth needs of GPUs while minimizing energy. Others[6] design a memory system that improves the energy efficiency of STT-MRAM as a main memory replacement. Awad *et al.*[7] propose designs for on-DIMM NVM controllers using software simulation. Contutto[8] is a platform to experiment with different memory technologies in an end-to-end system context.

A key aspect of the LiME design is to direct all memory references that are not satisfied in the cache to the PL.

Our paper studies the performance of a variety of data-intensive benchmarks by varying read and write latencies, ratios of read-to-write latencies, and degree of concurrency to characterize performance trends. In contrast to other studies, we use an abstract memory model agnostic to specific memory technology. Our model is parameterized by latency ranges and a bandwidth cap to uncover trends that characterize the performance of future memory systems.

Our contributions include quantitative assessment of performance at multiple ratios of cache to the main memory used as latency increases, the effects of concurrency levels over memory latencies ranging from 45 to 800 ns, and the effects of asymmetric read-to-write latencies. Our results show that effective use of cache can mitigate latency increases at a ratio of up to 1:2 between cache capacity and working set size. We find that concurrency is essential to improving performance as latency increases, but memory controllers will need to manage a larger number of outstanding memory requests to utilize available bandwidth.

EVALUATION METHODOLOGY

Our evaluations run application benchmarks under LiME. The LiME emulation framework [3] as shown in Figure 1 includes both hardware modules and software runtime libraries for the Zynq UltraScale+ device on a ZCU102 development board. This MPSoC contains a fixed logic processing system (PS) region and a programmable logic (PL) region. The PS contains a 1.2-GHz quad-core 64-b ARM Cortex-A53 CPU with a two-level cache hierarchy and 64-B cache lines. A 4-GB 64-b DDR4 memory is connected to the PS via a hard memory controller and used as the primary system DRAM. A second 512-MB 16-b DDR4 connects to the PL. We use this second DDR4 to optionally log memory request traces. FPGA hardware modules instantiated on the PL consist of a performance monitor to capture memory events, and a trace capture device to send memory traces to either the Trace DRAM or Ethernet. An optional accelerator block is included in the framework, but not used in this study. The delay units transmit external memory read and write transactions at specified latencies.

A key aspect of the LiME design is to direct all memory references that are not satisfied in the cache to the PL. This is accomplished by shifting the address of memory requests to a range mapped to the PL, which requires modifications to the firmware bootloader, the device tree, and the Linux build process. The memory requests flow through configurable delay units. Read and write delay are configured independently, enabling LiME to emulate asymmetric read/write latencies. The delay amounts are calibrated to account for CPU clock speed, the speed of the loopback circuitry in the PL, the bandwidth of the data channels, and latency within the PS memory controller subsystem. The multiple clock domains and datapaths associated with all of these components must be carefully configured

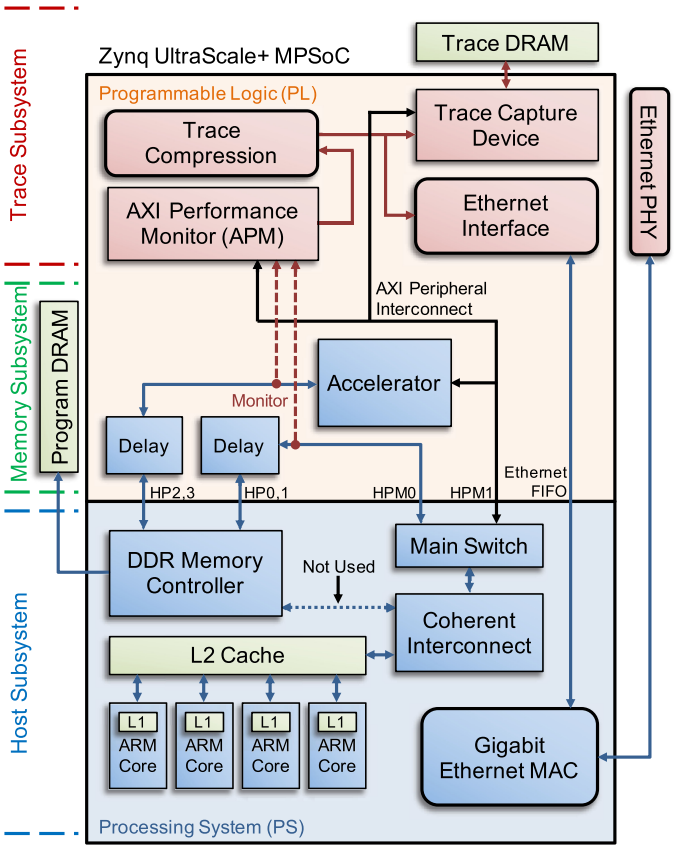

Figure 1. Zynq Ultrascale+ MPSoC with emulation framework.

to meet the desired emulated memory latency requirements.

As memory transactions travel to delay units, they are also routed to the AXI performance monitor (APM). The APM is configured dynamically at runtime to start or stop trace capture with a selective amount of detail. A trace fragment is shown in Table 1. The trace shows transactions of type read (R) or write (W). Other transaction types that are captured in a detailed trace include events for the first or last word in a data burst and the response. These events allow the response time and other metrics for a transaction to be calculated post capture. The address is a 64-b physical address. The AXI ID specifies the requester, which can be a core, the prefetch unit, or the cache. Time is shown in clock cycles, where each tick represents 0.16 ns. "RQ" is the outstanding request count, i.e., the number of pending memory requests. "LR" stands for the number of clocks since the last request on the same channel source:type:id. The final field is the latency of the request in clock cycles. In the emulated system, six clock ticks take 1 ns. The Latency column of the trace fragment shows that in this trace, the memory latency has been set to 800 ns (e.g., 4815/6 = 802.5).

EXPERIMENTS

Using LiME, we have conducted parameter sweeps over memory systems in which the main memory latencies model the range from fast DRAM to latencies of projected SCMs. Using the delay units we model latencies from 45 to 800 ns. Since the delay units provide separate read and write delay amounts, we can model asymmetric read/write latency. The emulator models a 2.75-GHz ARM quadcore A53 64-b CPU with 32-KB Level 1, 2-way set-associative instruction cache with parity (independent for each CPU); 32-KB Level 1, 4-way set-associative data cache (independent for each CPU); 1-MB 16-way set-associative Level 2 cache (shared between the CPUs); 4-GB main memory with latency modeled from 45 to 800 ns. The maximum emulated bandwidth is 44 GB/s. For most of the experiments, LiME runs the standard Linux Ubuntu distribution provided by Xilinx. The benchmarks are run single user to minimize noise. Since transparent huge page mode has a nondeterministic mix of 4-K and 2-MB pages, we chose 4-K pages for reproducibility of results.

The data-intensive benchmark set reflects likely use cases for SCM. Most of the benchmarks are read dominated. Since writing SCM consumes significant power and causes wear, write-heavy workloads such as logging or checkpointing are better suited to conventional DRAM. Two benchmarks feature balanced read–write workloads: STREAM triad, for read–write sequential access, and RandomAccess for read-modify-write random access. Matrix multiply, a ubiquitous kernel operation, is studied for both dense (DGEMM) and sparse (SpMV) matrices. The latter multiplies a sparse matrix with a dense vector,[9] and in our experiments, the matrix has 41 nonzeros per row in a banded diagonal pattern. Data-oriented benchmarks include Pagerank operating on a scale-free graph, KVQuery using the C++ standard template library associative map class on a bioinformatics data set, and

Published by the IEEE Computer Society

Table 1. Memory trace showing 64-B read and write accesses.

Type	Address	AXI ID	Time	RQ	LR	Latency
R	0x10BBF2D000	1069	22268	8	14268	4815
R	0x10BBFD040	1197	22572	7	14558	4809
R	0x10BBFD080	1325	22578	8	13546	4809
W	0x10FE576500	525	22702	1	8191	4812
R	0x105FE9A000	1453	23518	8	14119	4815
R	0x10BBF2D0C0	1581	23878	8	14344	4809
R	0x10A5E0ED40	1357	24014	8	14469	4814
R	0x104ABCDE00	1101	24042	8	13277	4819
R	0x10D2A33580	1261	25260	8	21059	4814
R	0x10A5EOED80	1229	27101	8	14188	4815
R	0x104ABCDE40	1485	27398	7	14789	4813
R	0x10D2A335C0	1389	27404	8	14496	4813
R	0x10D2A33600	1517	28351	8	13991	4809
R	0x102B41BE00	1645	28705	8	14322	4815
R	0x1076E384CO	1901	28846	8	25673	4814
R	Ox10D2A33640	2029	28879	8	28879	4809
W	0x102B484E00	525	29009	1	6307	4812
R	0x108E227300	1677	30092	8	24159	4815
W	0x102BB8D680	557	30220	2	30218	4812
R	0x1076E38500	1133	31934	8	16338	4815
R	0x108E227340	1805	32229	7	14794	4814
R	0x108E227380	1293	32235	8	14490	4813
R	0x108E2273C0	1037	33179	8	14488	4809
R	0x108E227400	1165	33538	8	14327	4809
R	0x10A5E0EDC0	1613	33677	8	19819	4814

ImageDifference on subsampled imagery which is used for satellite imagery change detection. These three well-known algorithms were coded in-house. A mini-application XSBench[10] represents "a key computational kernel of the Monte Carlo neutronics application OpenMC" and is part of the selection benchmark set for DOE supercomputer procurements. STREAM, RandomAccess, and DGEMM are from HPC Challenge Benchmark.[11]

As shown in Table 2, most of the benchmarks (Pagerank, KVQuery, XSBench, ImageDifference) are read-dominated (with a %load higher than 80%). RandomAccess has almost 50/50 load/store and STREAM has 60/40 load/store. RandomAccess has a very high cache miss rate, with 38% of cache accesses originating from the core missed by last level cache. At the other extreme, the cache miss rate is very low for ImageDifference as it can exploit spatial and temporal locality. From the fourth row, it is evident that several benchmarks (Pagerank, KVQuery, and XSBench) are wear-leveling friendly since they exhibit very low average write bandwidth

Table 2. Workload characteristics.

	Random Access	Pagerank	KVQuery	XSBench	STREAM	Image Difference	DGEMM	SpMV
%Load	53.6	93.8	81.6	76.9	60.1	60.1	87	97.9
%Cache misses	38.2	25.8	31.7	9.4	9.3	9.3	3.3	6.6
Average read BW (MBs)	696.3	641.7	707.4	778.1	3219.9	3219.9	1614.4	634.5
Average write BW (MBs)	388.5	6.9	36.9	3.7	2137.3	2137.3	194.6	36

demand to external memory (less than 37 MB/s). On the other hand, benchmarks like STREAM and RandomAccess, both of which have a more balanced read/write ratio, exhibit higher average write bandwidth demand. These statistics were obtained from single core runs in which the problem sizes were scaled to use the largest possible that could run on the ZCU102 development board.

Cache-to-Memory (C:M) Ratio

We approximate the effects of working set size relative to processor cache size by running different problem (and therefore dataset) sizes, thereby varying the C:M ratio of the benchmark. In this experiment, we contrast dense and sparse matrix multiplication to explore the extremes of cache friendly to random memory access patterns. Figure 2(a) and (b) shows performance profiles for the DGEMM and SpMV benchmarks. To study C:M ratios, different size matrices were used, so that the larger the matrix, the smaller the ratio of cache to matrix size in memory.

These sequential benchmarks were run on a single core in standalone mode in which a small support library is statically linked into the application program with no OS. The set of curves in the DGEMM plot show an inflection point between a C:M ratio of 1:2 and 1:4, indicating that for the small cache to memory ratios, the DGEMM kernel tolerates considerable main memory latency. This suggests that block methods typically used in matrix libraries that keep the working set in the cache can tolerate longer latencies. In contrast, SpMV shows a consistent and steady increase in execution time at each C:M ratio as latency increases. This trend was observed even for a 2:1 ratio in which the compressed format matrix should fit in cache, indicating matrix elements are not reused. For sparse operations, applications will need a high level of concurrency and greater throughput to offset performance loss at longer latencies.

Concurrency and Latency

To study the effects of concurrency, a set of multithreaded benchmarks are run. With a maximum of four cores, we vary concurrency from one to four threads using OpenMP parallelism. Each benchmark was configured with the largest possible problem size and all concurrency levels used that same problem size. The results are summarized in Figure 3. In these plots, the *x*-axis shows the emulated latency, and the *y*-axis is the emulated run time for the benchmark's region of interest. Each curve shows performance at a given concurrency level. Considering the large parameter sweep over various benchmarks, latencies, and concurrency levels, the benefit of FPGA emulation becomes apparent. As one example, the XSBench emulated runtime just for the region of interest was 2271 s. With the emulator's

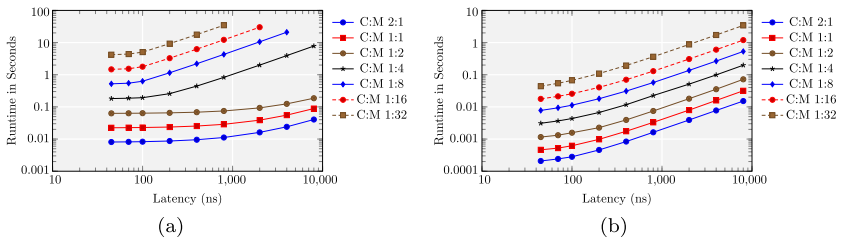

Figure 2. Execution time on 64-b processor at varying latencies and varying C:M ratios. (a) DGEMM. (b) SpMV.

Published by the IEEE Computer Society

20× slowdown, this took 12.6 h on the FPGA board. Software simulation at a factor of 10 000 over emulated time, would be prohibitive.

The results show that thread concurrency is beneficial, with a factor of two improvements in runtime going from a single thread to two threads, and another factor of two going from two to four threads.

The results show two categories of access patterns. Figure 3(a)–(d) has relatively random access and shows a consistent linear increase in runtime with increasing latency. This trend is reflected at each level of thread concurrency, indicating that for a given level of concurrency, higher latency results in lower performance. The reduced performance may be acceptable for read-dominated workloads with large data sets, offering a lower cost alternative to DRAM-only servers.

In contrast, the ImageDifference and STREAM benchmarks, both of which have regular access patterns, lose improvement from thread concurrency at four threads and higher latency. Figure 3 (e) and (f) shows that from 400 ns on, ImageDifference gets no benefit from four threads over three. STREAM loses thread concurrency benefit at 70 ns. A detailed study of the memory traces of these benchmarks was required to identify the bottleneck. As shown in Table 1, LiME can log each AXI memory-related transaction from the initial request to its completion. By keeping track of the number of in-flight memory requests, it was determined that at most eight memory requests could be pending (column "RQ" in the table). For ImageDifference and STREAM, there were almost always eight outstanding requests, while for the other benchmarks, that count varied over time but was mostly much less than eight.

In the Zynq UltraScale+ ARM microarchitecture, the LiME path through the PL to memory uses an eight entry deep FIFO, and once the FIFO is full, the CPU must wait for a pending request to complete before issuing another. With regular strided access, multiple threads plus the prefetch and write-back units were issuing read requests at a faster rate than the memory could service, causing a decrease in throughput. In fact, memory bandwidth reported by STREAM at high latencies was significantly less than at low latency. The implication of this finding is that memory controllers must support more in-flight requests to maintain throughput to effectively utilize higher latency memory systems. A standard DRAM memory controller approach such as coalescing requests may not be as effective in SCM when the workload accesses random addresses over a very large capacity memory. SSDs gain throughput

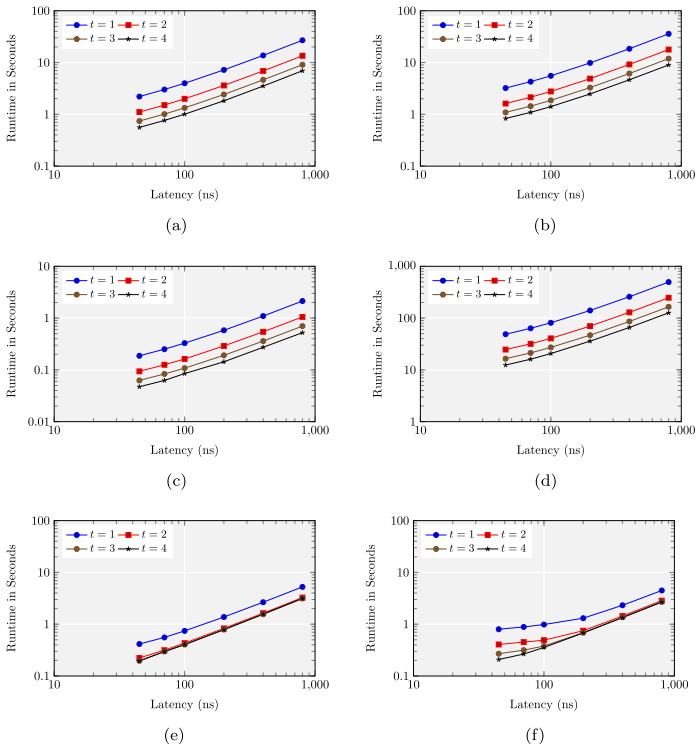

Figure 3. Execution time for application benchmarks at varying latencies for different number of threads. (a) RandomAccess. (b) Pagerank. (c) KVQuery. (d) XSBench. (e) STREAM-triad. (f) ImageDierence.

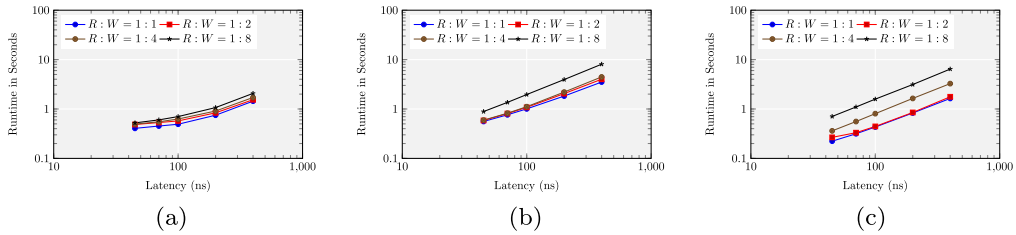

Figure 4. Execution time at varying R:W latency ratios. (a) ImageDifference (t = 2). (b) RandomAccess (t = 4). (c) STREAM-triad (t = 2).

improvement with high queue depth. For emerging memories with latencies approaching the storage tier, storage-tailored optimization may prove more effective than those used in traditional DRAM memory controller design.

Read/Write Latency Asymmetry

Many SCMs show higher write latency than read. For some memories, it is projected to be many factors greater. In this study, we sweep the read-to-write latency ratio parameter space to examine the impact of asymmetric read/write latency on performance. Figure 4 shows runtime profiles for different read latencies when the write latency is a multiple of the corresponding read latency. For example, at 200 ns on the *x*-axis, the R:W of 1:4 point shows execution time when writes take 800 ns. Three benchmarks are shown, the ImageDifference program with 3% writes, RandomAccess, with 12% writes, and STREAM, having 40% writes. The plots show that as the write fraction increases, the read/write asymmetry imposes an increasing cost on performance. The cost depends on the write fraction. For ImageDifference, only the 1:8 read/write latency ratio shows an appreciable slowdown. Random Access shows higher impact at the 1:8 ratio, with very little effect at lower ratios. For STREAM, the performance impact is considerable after 1:2. Reduced performance is correlated with reduced bandwidth.

The bandwidth reduction can come from multiple sources. For some SCMs, power constraints can throttle the memory system's write bandwidth. In our experiments, we did not change the memory's bandwidth cap. Our results show that at higher write latency, the CPU cores are stalled waiting for free cache lines, which are in turn dependent on cache line evictions and write back. Depending on cache capacity and workload, this effect may or may not overshadow the effect of reduced memory bandwidth. This finding indicates that write dominated applications such as checkpoints and log data may stall the entire server including other memory-bound traffic.

DISCUSSION

In this paper, we have exploited the LiME framework to sweep an extensive parameter space of emerging memories' latencies under a fixed bandwidth cap.

Key findings

- Caches can do well at hiding latency as long as the working data set size is within 2× of the cache size [see Figure 2(a)]. Note this ratio is for the cache to working set, not to full data set size. The matrix could be very large, but if processing is partitioned into cache-friendly blocks, performance can be maintained when memory latency increases.
- Applications with highly random access patterns not suitable for cache-friendly blocking slow down linearly with latency. For these applications, a throughput driven approach is desirable to hide latency with concurrent data parallel memory accesses. For example, in Figure 3(c), using three threads with a 200-ns memory gives equivalent performance to two threads at 100 ns.
- At higher latencies and with higher concurrency, the number of pending memory requests becomes the limiting factor to performance. As memories come closer to storage-like latencies, memory controllers can benefit from SSD-like optimization techniques such as high queue depth.
- None of the experiments were able to use the available emulated 44-GB/s bandwidth. At each concurrency level, as latency increased, the application's bandwidth demand decreased, and that decrease tracked the application's performance decrease. As concurrency increased, bandwidth increased up to the limit of the maximum number of pending memory requests.
- For asymmetrical read/write latency ratios, the increase in runtime tracked the write fraction. At 40% writes, a latency ratio of more than 1:2 had a significant impact. For lower write fraction, a latency ratio over 1:4 showed performance loss. These results quantitatively validate the intuition that large capacity, high write latency SCMs are not well suited to write dominated workloads.

Limitations

The LiME framework has allowed us to measure performance very accurately and run a large set of studies quickly compared to software simulation. However, the emulator has a

Published by the IEEE Computer Society

fixed framework of CPU and cache hierarchy, and studies with other processor/cache models can provide additional insights. Our study has used a simple cache hierarchy of SRAM caches and main memory and studied benchmarks that either can exploit the processor cache (DGEMM-like) or have data sets and access patterns that are unlikely to benefit from the cache. Using additional levels of cache, including a DRAM cache for large capacity persistent memories, may mitigate performance impact at higher latencies for applications that can still benefit from cache but have a larger working set than fits in CPU SRAM cache. This study used a simple abstract model of the memory system with a fixed latency. Incorporating distributions rather than average latency will more accurately model workload performance as a function of both access patterns and the internal memory system architecture.

This study has focused on speed performance; however, energy is as important a factor. Memory system power models must be included in the analysis to understand the energy usage. The memory capacity also plays a factor in an energy analysis. While it has been shown through simulation that using persistent memory may increase an application's energy usage, for large enough capacity and periods without use, a server with persistent main memory may still offer energy savings over an all-DRAM system. Wear is also a concern for write-heavy workloads using persistent memory. On-going research into wear-leveling techniques, combined with DRAM caching will push the frontier on this challenge.

SUMMARY

The emergence of novel memory technologies is one of the most exciting and fast-moving research topics in computer architecture. Even as memories such as 3-D XPoint emerge, there are many unknowns in the latency/bandwidth/capacity space. Understanding the benefits and drawbacks of these memories from an application performance perspective helps memory system designers and applications developers alike. The goal of our LiME tool development and this study is to provide quantitative performance data in the latency parameter space. Our findings highlight the increasing importance of effective cache utilization, the need for concurrency both within the application and within the memory controller, and the performance impact of asymmetric read/write latencies.

ACKNOWLEDGMENT

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 and was supported by LLNL LDRD Project 16-ERD-005. LLNL-JRNL-754877.

REFERENCES

1. F. T. Hady *et al.*, "Platform storage performance with 3D XPoint technology," *Proc. IEEE*, vol. 39, no. 10, pp. 1822–1833, Sep. 2017.
2. J. S. Vetter *et al.*, "Opportunities for nonvolatile memory systems in extreme-scale high-performance computing," *Comput. Sci. Eng.*, vol. 17, no. *2*, pp. 73–82, Mar. 2015.
3. A. K. Jain *et al.*, "Microscope on memory: MPSoC-enabled computer memory system assessments," in *Proc. IEEE Symp. Field-Program. Custom Comput. Mach.*, 2018, pp. 173–180.
4. H. Volos *et al.*, "Quartz: A lightweight performance emulator for persistent memory software," in *Proc. 16th Annu. Middleware Conf.*, 2015, pp. 37–49.
5. M. O'Connor *et al.*, "Fine-grained DRAM: energy-efficient DRAM for extreme bandwidth systems," in *Proc. IEEE/ACM Int. Symp. Microarchit.*, 2017, pp. 41–54.
6. E. KItrsay *et al.*, "Evaluating STT-RAM as an energy-efficient main memory alternative," in *Proc. Int. Symp. Perform. Anal. Syst. Softw.*, Apr. 2013, pp. 256–267.
7. A. Awad *et al.*, "Performance analysis for using non-volatile memory DIMMs: opportunities and challenges," in *Proc. Int. Symp. Memory Syst.*, 2017, pp. 411–420.
8. B. Sukhwani *et al.*, "Contutto: A novel FPGA-based prototyping platform enabling innovation in the memory subsystem of a server class processor," in *Proc. IEEE/ACM Int. Symp. Microarchit.*, 2017, pp. 15–26.
9. H. Gahvari *et al.*, "Benchmarking sparse matrix-vector multiply in five minutes," in *Proc. SPEC Benchmark Workshop*, Jan. 2007.

10. J. R. Tramm *et al.*, "XSBench—The development and verification of a performance abstraction for Monte Carlo reactor analysis," in *Proc. PHYSOR 2014 – Role Reactor Phys. Toward Sustain. Future*, 2014.
11. HPC Challenge Benchmark. Available at: http://icl.cs.utk.edu/hpcc/

Abhishek Kumar Jain is a Postdoctoral Research Staff Member with the Center for Applied Scientific Computing, Lawrence Livermore National Laboratory, Livermore, CA, USA. His research interests include reconfigurable computing, high-performance accelerators, and data-intensive computing architectures. He received the Ph.D. degree in computer engineering from Nanyang Technological University, Singapore, in 2016. Contact him at jain7@llnl.gov.

Scott Lloyd is a Computer Scientist with the Center for Applied Scientific Computing at Lawrence Livermore National Laboratory, Livermore, CA, USA. He cofounded a parallel-processing company (which became Linux Networx), worked with diagnostic imaging at GE Medical Systems, and investigated processor-in-memory architecture at Micron Technology. His research interests include high-performance computing, computer architecture, and reconfigurable computing. He received the Ph.D. degree in computer science from Brigham Young University, Provo, UT, USA in 2011, and has broad experience in industry. Contact him at lloyd23@llnl.gov.

Maya Gokhale is a Distinguished Member of Technical Staff with the Lawrence Livermore National Laboratory, Livermore, CA, USA. Her career spans research conducted in academia, industry, and National Laboratories. Her research interests include data intensive architectures and reconfigurable computing. She received the Ph.D. degree in computer science from University of Pennsylvania, Philadelphia, PA, USA, in 1983. She was the corecipient of an R&D 100 award for a C-to-FPGA compiler, corecipient of four patents, and coauthor of more than one hundred technical publications. Contact her at gokhale2@llnl.gov.