SAND2017-4859C

Performance Analysis for Using Non-Volatile Memory DIMMs: Opportunities and Challenges

ABSTRACT

DRAM scalability is becoming more challenging, pushing the focus of the research community towards alternative memory technologies. Many emerging non-volatile memory (NVM) devices are proving themselves to be good candidates to replace DRAM in the coming years. For example, the recently announced 3D-XPoint memory by Intel/Micron promises latencies that are comparable to DRAM, while being non-volatile and much more dense. While emerging NVMs can be fabricated in different form factors, the most promising (from a performance perspective) are NVM-based DIMMs. Unfortunately, there is a shortage of studies that explore the design options for NVM-based DIMMs.

The non-deterministic latencies of NVMs, due to read/write latencies and power consumption asymmetry, in addition to the limited write endurance, which requires wear-leveling techniques, require a specialized controller. The fact that future on-die memory controllers are expected to handle different memory technologies pushes future hardware towards on-DIMM controllers. In this paper, we propose an academic design for NVM-based DIMMs with internal controllers, explore their design space, evaluate different optimizations and reach out to several architectural suggestions. Finally, we make our model publicly available and integrate it with a widely used architectural simulator.

1 INTRODUCTION

Scaling the cell size of DRAM is becoming more challenging [17] and emerging NVMs are being widely studied as potential replacements for DRAM. Emerging NVMs, such as Phase-Change Memory (PCM), have latencies that are comparable to DRAM (less than order of magnitude slower) and promise high densities [20, 14, 23, 7, 16]. Additionally, emerging NVMs are non-volatile, which eliminates the need for refresh power. On the other hand, emerging NVMs have limited write endurance and high write latencies.

While NVM technologies are expected to appear soon in the market [3], the parameters, such as read/write latencies and power limitations are expected to change as the technology matures. While NVMs are very attractive for adoption in large scale HPC systems, it is unclear how they will affect the performance. For instance, how would the write latency affect the performance, *i.e.*, how sensitive are HPC applications to write latency? How effective are write latency mitigation techniques, such as write-cancellation, for HPC applications? Another example, given the high read latency of NVMs compared to DRAM, how effective are row buffers? Can row buffers effectively mask the high latency of emerging NVMs?

Most previous research work selected specific NVM parameters and used them without considering the potential impact on performance. This is critical, especially for HPC systems where multiple vendors are able to provide the memory technology, often times with different memory characteristics. As such, it is important to know what parameters are critical for performance and power while considering the price.

Many of the design parameters of NVM can impact the performance of systems, particularly at scale. Examples are the read latency from NVM cells, write latency and its associated optimizations such as write-cancellation, the maximum number of concurrent writes (typically limited by power budget), the maximum number of outstanding requests, number of banks, number of channels, internal caching and scheduling. In this paper, we propose and describe in detail an architectural simulation model for NVM-based DIMMs. We use our model to evaluate the impact of different optimizations and parameters on the performance of memory systems built from NVM-based DIMMs. We integrate our model with the Structural Simulation Toolkit (SST) simulator [21], and make it publicly available.

Our focus in this paper is on memory-intensive HPC applications. Our analysis shows that some applications are very sensitive to NVM read latency. We also observe that using large row buffers does not help for most of the studied applications. Moreover, our study shows that some applications are very sensitive to the NVM write latency. The maximum number of allowed concurrent write operations, which is limited by the power budget, can significantly affect the performance of several applications. Additionally, we examine state-of-the-art write latency mitigation techniques, such as Write-Cancellation [18], and examine their sensitivity to write latency and impact on performance. Finally, we study the potential performance impact of on-DIMM caching and compare it with DRAM-only system.

The rest of the paper is organized as follows. First, in Section 2, we discuss the key characteristics of emerging NVM technologies. In Section 3, we describe an open-source architectural simulation model for NVM-based DIMMs, its key parameters and the rationale behinds each of them. In Section 4, we describe our evaluation

Conference'17, Washington, DC, USA
© YYYY ACM. 978-x-xxxx-xxxx-x/YY/MM...$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

methodology, including the default parameters. Section 5 includes a detailed analysis of the performance sensitivity for different NVM parameters. Section 6 discusses the conclusions from Section 5 and propose several architectural optimizations. Finally, in Section 7, we conclude our work with suggestions for future work.

2 BACKGROUND

In this section, we discuss emerging non-volatile memory technologies, their key characteristics and design options.

2.1 Emerging Non-Volatile Memory (NVM) Technologies

Emerging Non-Volatile Memory (NVM) technologies, such as Phase-Change Memory (PCM), Memristor and Spin-Transfer Torque RAM (STT-RAM), have different characteristics. Some of these technologies have high densities, *e.g.*, PCM and Memristor, making them very promising candidates for building main memory and storage. Others, such as STT-RAM, have low density, making them more appealing for building Last-Level Caches (LLCs). One common feature across these technologies, is non-volatility, which can be defined as the ability to retain memory cells values even when there is no power supplied. However, issues such as resistance drift may require refreshing cells, but with considerably lower frequency than current DRAM technology.

Across emerging NVM technologies, PCM is thought to be very promising for replacing DRAM. PCM has much higher density, which promises large capacity main memories, it also has read latencies that are comparable to DRAM. Unlike DRAM, PCM's non-volatility feature eliminates the need for expensive refresh power, resulting in near-zero idle power. On the other hand, PCM has long write latencies, which can go up to thousands of cycles, and limited write endurance. In this paper, we focus on the usage of PCM technology as main memory.

2.2 Designs and Optimizations

Since emerging NVMs are still in their infancy, there is still no clear answer of how exactly they should be designed for high-performance compute nodes. For instance, some vendors might prefer to offload scheduling, optimizations, such as wear-leveling and reduction, to occur at an ***on-die (near processor) memory controller***. While this approach can bring down the cost of NVM-based DIMMs, it comes at the cost of compatibility with different memory technologies and extra on-chip area that might be never used, *e.g.*, for systems which do not use NVMs. Furthermore, the non-deterministic timing of NVMs renders processor-side access protocols difficult to implement. An alternative design option is an ***on-DIMM controller (near memory)***. This approach requires a packet-based protocol, where the processor-side memory controller sends a packet with a request ID, and command information, then the on-DIMM controller decodes the command and executes it. Optimizations, such as wear-leveling, write-cancellation, internal scheduling and power-limiting considerations, are implemented on the DIMM, *i.e.*, by the memory vendor. This approach is promising for several reasons. First, it does not expose the internal characteristics and design details to processor vendors, while still implementing appropriate optimizations through the internal controller. Second, it requires negligible modifications on the processor side, to enable the integration of new technologies. Finally, due to signal integrity challenges, buffering commands and data internally is easier to implement, which is becoming more common for DRAM, e.g., LRDIMM and RDIMM[15, 1, 2].

In this paper, we adopt the latter design approach, which we expect to be the most dominant for future memory systems. Our expectation is based on designs appearing in industrial patents and current prototypes [4, 5]. Figure 1 shows an example design of near-memory controllers.

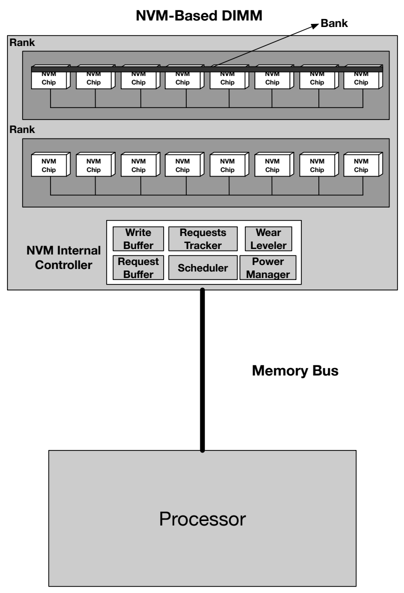


Figure 1: Example of near-memory (on-DIMM) controller, similar to [4, 5]**.**

The internal controller typcially includes optimizations for writes, wear-leveling, internal caching and buffering, power managment and scheduling. One promising technique that has been shown to improve performance is write-cancellation[18]. Write-cancelation cancels pending write operations in order to service read operations, which are usually on the critical path, avoiding a long time delay for read operations. For wear-leveling, Start-Gap wear leveling technique is expected to be used [19]. PCM write operations incur high power, which necessitates a power management scheme to limit the number of concurrent NVM device writes to avoid exceeding the power budget of the DIMM.

3 THE MESSIER NVM MODEL

In this section, we describe our NVM software model, which we use to conduct our experiments and base our analysis upon.

Our NVM model assumes a DIMM that consists of one or more ranks, where each rank consists of several banks. Each bank has a fast row buffer that caches the most recently read row. Note that since PCM writes must persist, PCM bypasses the row buffer and writes directly to the NVM cells in this case. The DIMM has an internal memory controller that tracks outstanding requests, schedules requests and implements several optimizations, such as write cancellation and caching. If the request is a write operation, once scheduled, it will be written to the write buffer, which is guaranteed to be persistent, either through a small capacitor to power performing a back-up or being fast NVM memory such as STT-RAM. Figure 2 shows a high-level description of our software model.

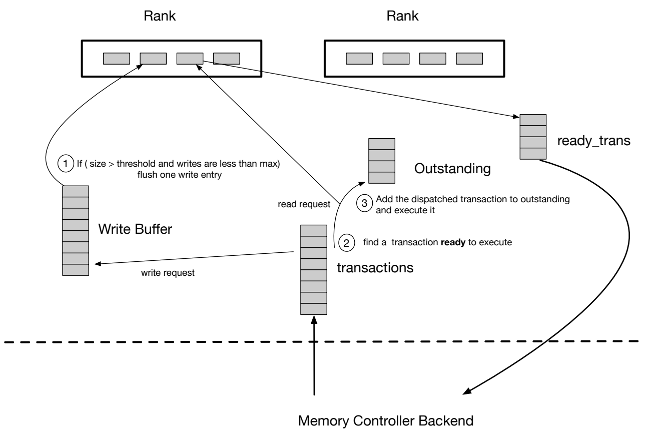

Rank
Rank
ready_trans
Outstanding
1 If (size > threshold and writes are less than max) flush one write entry
read request
3 Add the dispatched transaction to outstanding and execute it
Write Buffer
2 find a transaction **ready** to execute
transactions
write request
Memory Controller Backend

Figure 2: The software model for our NVM-based DIMM model.

To better describe the parameters, we will go through a read request scenario. Once a read request is received, it is placed in the transaction queue, where it waits until being scheduled. Once the scheduler decides to dispatch the request, which only happens if the corresponding bank is free and the rank internal circutary/bus/channel is free, it checks if it is a row buffer hit or miss, and accordingly issue a command to the corresponding NVM bank. Sending the command occupies the rank bus for `tCMD` cycles. Once the command is received by the bank, if row activation (in case of row buffer miss), it will take `tRCD` cycles to load the data into the row buffer. Later on, after the row activation or in case of row buffer hit, the scheduler will again send another command to read the data from the bank, which will only happen when the rank bus is free. Once the command is received by the bank, it occupies `tCL` cycles to read a column and `tBURST` cycles to transfer the data over the bus. The request will be buffered in the `ready_trans` buffer before the data is sent back to the processor and notifying the on-die memory controller of the request completion.

In case of a write operation, once the request is scheduled, the data will be written to the persistent write buffer, and immediately notify the on-die memory controller of the request completion. Note that a write is scheduled only when the write buffer is not full. To avoid throttling the system as the write buffer is nearing getting full, a flushing mechanism is deployed. Typically, a threshold value is deployed to determine when to start flushing the write buffer, through prioritizing evicting write entries over servicing new requests. The maximum number of concurrent writes is limited by `max_writes` parameter, which can be set based on the power budget and thermal limitations. When a write is evicted, it occupies the bank for `tCL_W` cycles, while the rank bus is occupied for the time of sending the data and write command, `tBURST` and `tCMD`, respectively.

4 METHODOLOGY

We use the Structural Simulation Toolkit (SST) [21] to conduct our experiments and analysis. SST was configured to model 8 cores with private L1 caches and L2 caches. The L3 cache is shared across all cores and paritioned into eight banks. Our simulation default parameters are shown in Table 1. The simulation source code for these experiments is available on the SST repository[1]. The simulation infrastructure allows us to model the entire cache hierarchy, coherency, and the main memory latencies. SST's memHierarchy, Merlin, and Ariel components were used for the caches, on-chip network, and processors respectively.

Table 1: Simulation Parameters

Component	Parameters
Core	2GHz, 3 issue / cycle, 16 max. outstanding memory requests
Coherency	MESI protocol
L1	32KB, 8-way, 64B cache line, 4 cycles
L2	256KB, 8-way, 64B cache line, 6 cycles
L3	16MB, 16-way, 64B cache line, 12 cycles
Memory	four channels, one DIMM per channel
NVM DIMM	2GHz clock, 32 banks, 32 outstanding, 32 write buffer size, max of 4 concurrent writes
NVM DIMM Timing	tRCD=150ns, tCL_W=500ns, tCL=15ns

Since our focus is HPC appications, we use five Miniapps from the U.S. Department of Energy (U.S. DoE): miniFE [11], an unstructured implicit finite element code; Lulesh [13, 12], a hydrodynamics code; Pennant [8], an unstructured mesh physics mini-app; SimpleMoC [9], a mini-app to study Method of Characterstics (MoC) for 3D neutron transport calculations; and XSBench [22], A mini-app that represents a key computational kernel for the Monte Carlo Neutronics application OpenMC. Each application was executed

[1]https://github.com/sstsimulator/sst-elements/tree/ ADVANCED_MESSIER

with 8 threads, starting from the region of interest until at least one core executes 100M instructions (8̃00M instructions total). Additional parameters are listed in Table 2.

These applications were selected because they are memory-intensive and exhibit a diverse set of main memory access patterns.

Table 2: Application Parameters

Application	Options
miniFE	`-nx 140 -ny 140 -nz 140`
Lulesh	`-s 120`
Pennant	`leblancbigx2.pnt`
SimpleMoC	`-t 8 -s`
XSBench	`-s large -t 8`

5 DESIGN SPACE EXPLORATION

In this section, we investigate the impact of several state-of-the-art optimizations and how varying several NVM parameters can affect the performance.

5.1 The Impact of Write Latency and Write Cancellation

Write latency is considered to be one of the key challenges for using emerging NVMs as main memory. So, we begin our design exploration by studying the sensitivity of the write latency of NVM devices. Figure 3 shows the impact of write latency on performance. We vary the write latency, `tCL_W`, from 100 to 1000 cycles in 100-cycle increments. As expected, for most of the applications, the performance decreases as the write latency increases. The exception here is XSBench, which doesn't have very many writes.

One way to combat the the impact that write latency can have on application performance is to use write-cancellation, as described in Section 2.2. From Figure 3, we can observe that at low write latencies write-cancellation can actually hurt the application performance. This is because it can increase the average number of cycles that a bank is allocated for a write operation without actually decreasing the read latency as intended. The implementation of write-cancellation for this study uses adaptive thresholds[18]. This adaptive technique uses the elapsed time since the beginning of the write as well as the current number of entries in the write buffer to determine whether or not to cancel the write. The rationale behind this implementation is to achieve a good balance between not aggregating too many writes while still using write cancellation effectively.

5.2 Power Constraints for Concurrent Writes

NVM write operations involve applying high current to change a cell state. However, due to cooling constraints and thermal limits, a maximum power budget is given for each DIMM. Accordingly, to abide by that power budget, each DIMM should limit the maximum number of concurrent writes to the NVM banks. Limiting this number to only few concurrent writes will increase the chances of filling the write buffer, placing back pressure into the memory system. In contrast, allowing a large number of concurrent writes may cause the system to exceed its given power budget. Figure 4 shows how the number of concurrent writes affects the overall execution time of selected applications.

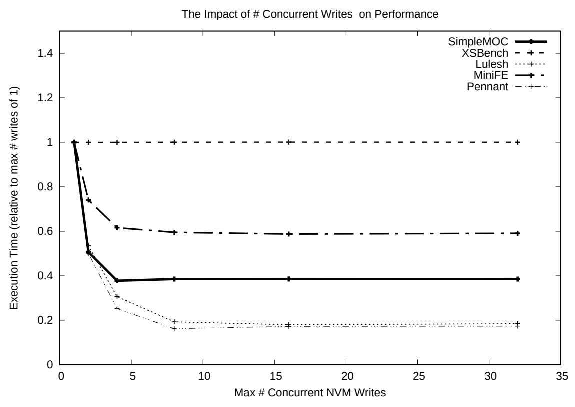


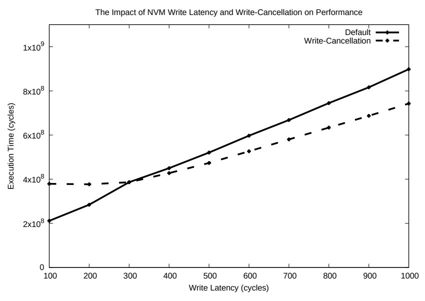
Figure 4: The impact of maximum number of concurrent writes on performance.

From the figure, we can observe that some applications, such as Pennant and Lulesh, are very sensitive to this parameter. This is consistent with our findings from Section 5.1, where we observed similar sensitivity for the write latency. On the other hand, we can observe that some applications, such as XSBench, have negligible sensitivity to the number of concurrent writes due to the read/write patterns inherent in the application.

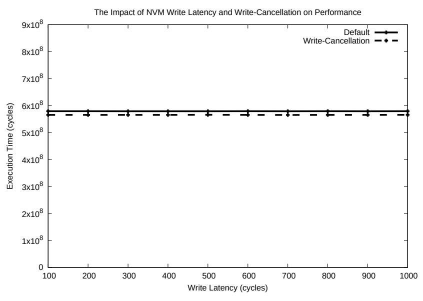
5.3 NVM Read Latency

While NVM read latency is much better than that for writes, it is still slower than that of DRAM. To study effect this has on application performance, we vary the NVM read latency, i.e., `tRCD`, and observe the change in the execution time, as shown in Figure 5.

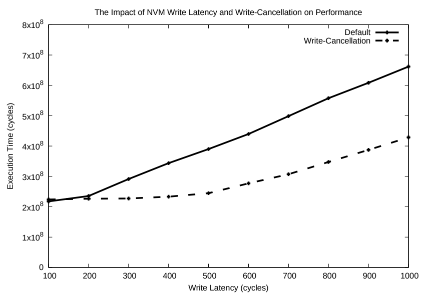
We can observe that some applications are highly sensitive to read latency, while others are not. Specifically, we can observe that Lulesh and Pennant are minimally affected by increasing the read latency, which can be explained by our observations on Section 5.1; Lulesh and Pennant performance is heavily dominated by the write latency.



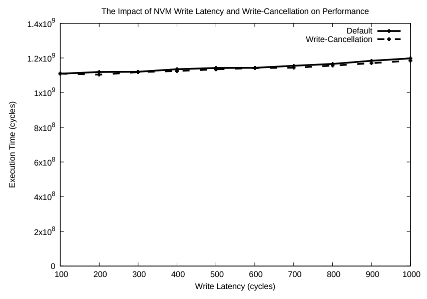
(a) Lulesh



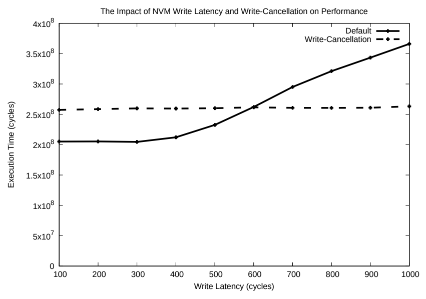
(b) XSBench



(c) Pennant



(d) MiniFE



(e) SimpleMOC

Figure 3: The impact of write latency along with and without write cancellation on performance

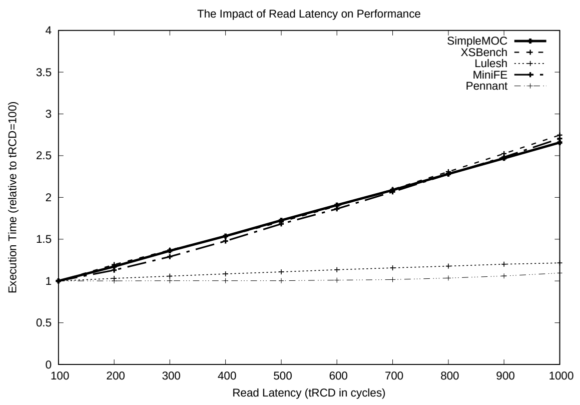


Figure 5: The impact of NVM read latency on performance.

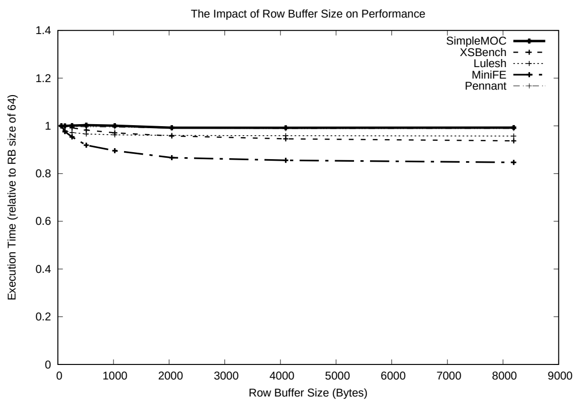


Figure 6: The impact of row buffer size on performance.

5.4 Row Buffers Locality

As the read latency can have significant impact on the performance of some applications, we now explore a way to mitigate it. A common way to mitigate high read latency is through row buffers, which cache the row of the most recently accessed cache line in a bank. To study the effectiveness of this technique, we vary the row buffer size from 64B to 8KiB, as shown in Figure 6.

We can observe that applications like XSBench and MiniFE benefit well from increasing the row buffer size, however, some applications are less sensitive to row buffer size, *e.g.*, SimpleMoC and Pennant.

5.5 The Impact of Internal Caching on Performance

One way to improve NVM performance is through caching blocks internally. This internal cache is checked in parallel when adding the request to the transactions queue. If the block is found, *i.e.*, a cache hit, the block will be returned from the cache and the pending request will be squashed. As the NVM does not incur significant idle power, the additional power overhead of SRAM or DRAM caches can still be comparable to DRAM-only systems. To study the performance gains of caching, we model an internal cache inside each DIMM with an access latency of 15 cycles.

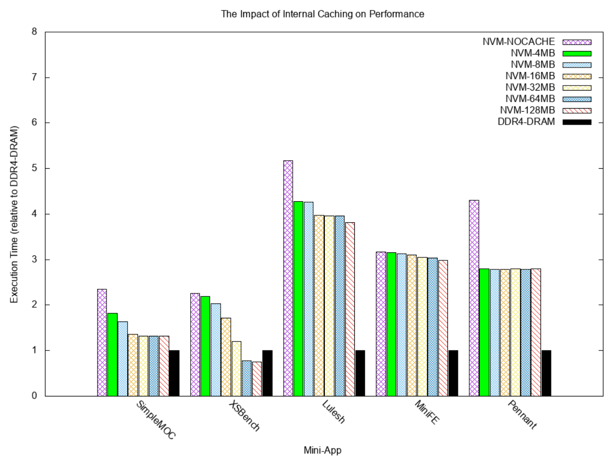


Figure 7: The impact of internal caching on performance.

Figure 7 shows the results for using caches and compare it with no-cache-NVM and DRAM-only systems. We can observe that most applications benefit from using a cache as small as 4MB. However, even with large caches, the NVM performance is by far worse than a DRAM-only system.

5.6 Paged Multi-Level Memory

Another mechanism to improve NVM performance is a multi-level memory (MLM). In this organization (See Figure 8), main memory is comprised of both NVM and DRAM memories. Memory is accessed through a controller that can implement a number of policies to determine which data is placed in the fast, stacked DRAM or in the slower NVM-based DIMMs. An SRAM table within the controller contains the mapping of which pages are in which memory and additional meta-information (e.g. page access frequency) to implement its paging policy.

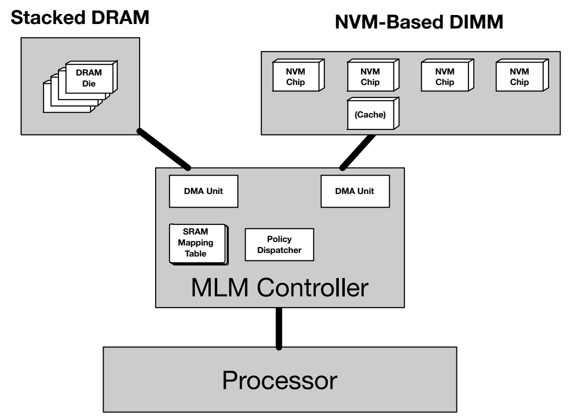


Figure 8: Multi-Level Memory Organization

There are several possible policies for MLM management[10] which govern which pages are removed from fast memory and which are added. For this work, we tested[2] the addMFRPU (More Frequent, More Recent Previous Use with threshold) and addT (simple threshold) addition policies and a simple LRU (Least Recently Used) replacement policy. We found the addMFRPU yielded better performance on XSBench and Lulesh, however its performance was no more than 1-3% better than the simple addT policy. More significant was the threshold level. The threshold level defines a minimum number of accesses to a page before the page is considered for addition to the fast memory. We tested two thresholds (2 and 16) and found that different applications benefit form different thresholds.

Figure 9 summarizes the results of different policies for an MLM system with roughly $\frac{1}{4}$ of the memory as fast DRAM and the remainder NVM and using the addMFRPU policy. We varied both the threshold and the presence of a 16MB cache in the NVM. The results are very application dependent. XSBench did better with a high threshold, while Lulesh, MiniFE, and Pennant do worse with a high threshold and no cache, but prefer a high threshold if there is a cache. Generally, an NVRAM cache did not help the performance of an MLM system, as would be expected since the page-level caching of the MLM system would interfere with the block-level cache of the NVM cache. However, the best XSBench performance was achieved with both paged-level MLM caching and the NVM cache. In general, performance was less than that of DRAM, though SimpleMOC performance was as good or better.

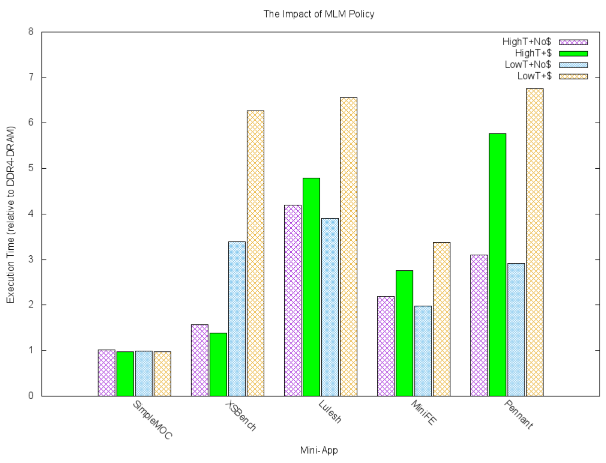


Figure 9: MLM Paging policy impact (lower=faster)

We also examined the impact of amount of fast stacked DRAM on the application performance (Figure 10). In all cases, total main memory was 1GB. Lulesh, MiniFE, and SimpleMOC were largely insensitive to the size of the “fast” memory. XSBench and Pennant

[2]https://github.com/sstsimulator/sst-elements/tree/afrodri/pagedMessier commit 3026e21

were very sensitive with XSBench more than doubling performance.

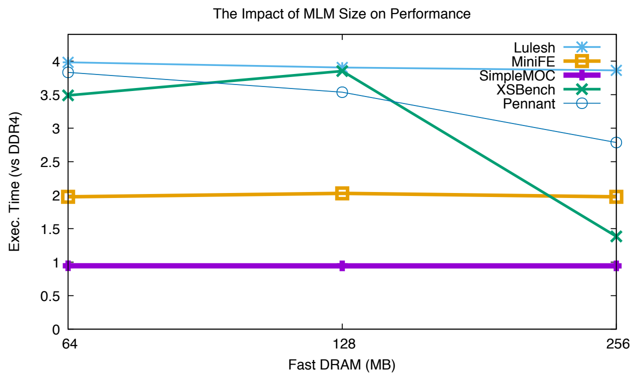


Figure 10: MLM "Fast" memory size vs. Performance (lower=faster)

Overall, a MLM organization shows promise in improving the performance of a NVM system, however in most cases the raw performance is still inferior to conventional DDR DRAM.

6 DISCUSSION

6.1 Performance

From Section 5, we can observe that the write latency of emerging NVMs can have a large impact on application performance. Although the write-cancellation technique may reduce much of the write overhead, some applications, such as Lulesh and Pennant, still suffer from long write latencies. Given the power limits of concurrent writes, we could also observe how this significantly affects the performance. Based on our observations, we can conclude that NVMs with long write latencies, if used as main memory, can incur significant overhead. While our results raise a warning for using emerging NVMs as the sole building block of the main memory, it helps to provide a case for architectures with multi-level memory – where NVMs can be used as an extension to memory capacity[6]. Additionally, we found that internal caching within NVM-based DIMMs is of limited use, which raises the case for software-managed caching for hot pages. For energy efficiency, we found that some applications do not benefit as much from large row buffer sizes, which motivates dynamic enabling/disabling or adjustable size row buffers solutions.

6.2 Cost & Performance

Even with caching, NVM main memories generally have lower performance than conventional DRAM memories. However, the value proposition of NVM is not raw performance but its potential cost and power savings. Current and emerging NVM technologies have storage densities much higher than conventional DRAM cells, which will lead to significant cost savings. An NVM main memory may not be higher performance, but with its much lower cost it may still be a valuable architectural alternative.

To test this, we propose a simple cost model (Table 3) based on rough cost per bit for different memory technologies. These cost estimates are based on the relative silicon area, or (in the case of Stacked DRAM) an adjustment for higher packaging costs. Though these numbers are open to debate, they provide a useful starting point for cost-performance analysis.

Table 3: Cost Model

Memory	Cost/Bit	Use
DDR4	1.0	Baseline Configuration
Stacked DRAM	1.25	"Fast" MLM
SRAM Tags	22.0	Storage for MLM meta-data
SRAM Cache	20.0	NV-DIMM Cache
NVRAM	0.133	NV-DIMM

Using this simple cost model, Figure 11 shows cost and performance points for a variety of memory configurations – conventional DDR4 (DDR4); NVM possibly with an internal SRAM Cache (NV+(Cache)); and paged NVM with a stacked DRAM page cache (NV+DRAM). With the exception of XSBench, NVM systems have lower performance. However, many of the NVM systems are much lower cost. Depending on the goals of the system designer, there are many cases where an NVM-based main memory system make sense.

Examined another way, Figure 12(a) shows the raw performance of the best configuration (cache size, paging policy, etc...) for the paged NVM system and for NVM systems with and without internal caching. With a few exceptions (SimpleMOC with 256MB of stacked DRAM, low threshold addT policy, NVM internal cache and XSBench NVM with 128MB caches), performance is worse than a conventional DDR4 system. However, Figure 12(b) shows that the performance/cost ratio for NVM systems can outperform DDR-based systems for all applications.

7 CONCLUSIONS

In this paper, we propose an new architectural simulation model for NVM-based DIMMs and explore implementation options. This model is highly parameterized and provides fast execution performance to permit scaling for long-duration simulations or complex application modeling. Later, we use the model to explore performance sensitivity to different NVM parameters. We used our model to investigate key parameters such as read latency, write latency, number of concurrent writes, row buffer size, internal caching, the effectiveness of the write-cancellation technique, and integration with paged multi-level memory systems.

Our study showed that the studied HPC applications vary in their performance sensitivity to read latency. For instance, we

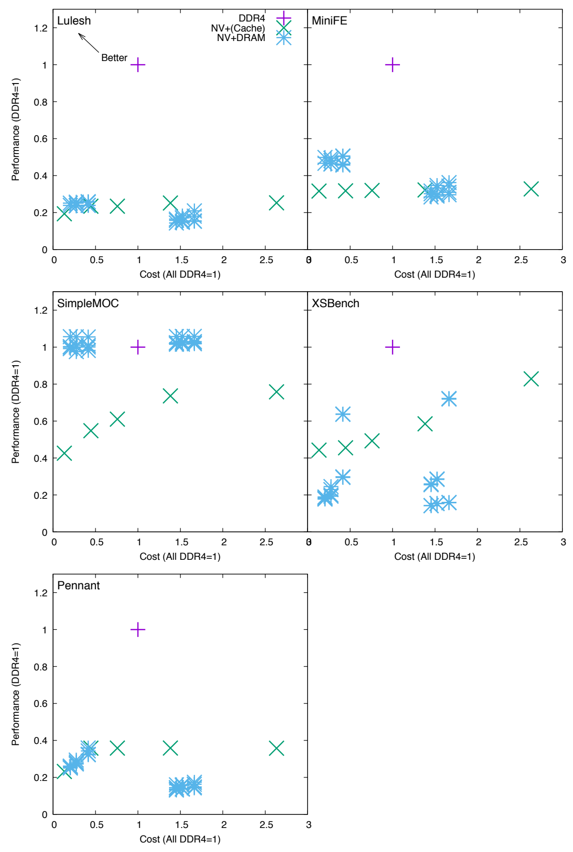


Figure 11: Cost and Performance Tradeoffs

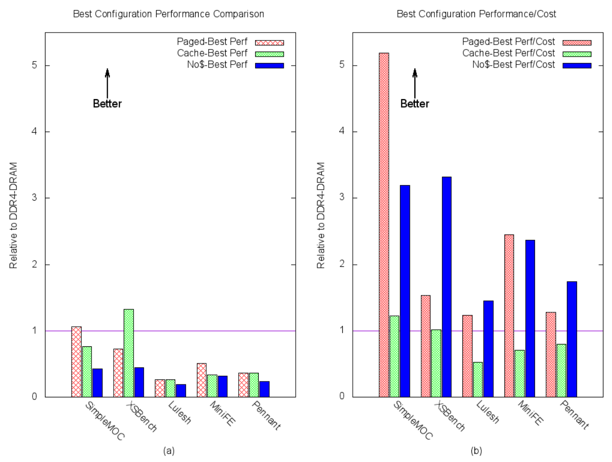


Figure 12: perf-cost

found that MiniFE and SimpleMoC are very sensitive to read latency, while less sensitive to write latency. In contrast, we found that MiniFE and SimpleMoC has less sensitivity to write latency, when compared to Pennant and Lulesh. We also studied how limiting the number of concurrent writes can affect the performance. Our analysis also shows the potential gains for increasing the row buffer sizes and augmenting DIMMs with internal caching.

We publish our infrastructure integrated in a widely used simulator (SST), to enable community researchers to investigate designs such as hybrid memory systems, performance optimizations and write latency mitigation optimizations.

These experiments show that NVM-DIMM based systems generally have lower performance than conventional DDR4 systems. However, when analyzed with memory system cost in mind, main memory with NVM becomes more attractive. There are a number of configurations which provide a better performance / cost tradeoff than conventional DDR-based main memory.

These results can be used to guide future NVM-DIMM implementations and can be used by system architects to select a more efficient memory system.

For future work, we plan to incorporate models for Multi-Level Cell (MLC) technologies and model the impact of read latency asymmetry for different levels in cells. We also plan to investigate the impact of wear-leveling techniques, such as start-gap, in the lifetime of the system. Additional MLM policies can be crafted for NV memory, particularly policies that account for the difference between read and write latency. The impact of power and energy on total system cost will also be explored.

REFERENCES

[1] 4RCD0124K DDR4, http://www.idt.com/.
[2] CAB4A DDR4, http://www.ti.com/lit/ds/symlink/cab4a.pdf.
[3] Intel 3D XPoint.
[4] Published U.S Patent Application. Dynamic partial power down of memory-side cache in a 2-level memory hierarchy, PCT/US2011/066302.
[5] A. Akel, A. M. Caulfield, T. I. Mollov, R. K. Gupta, and S. Swanson. Onyx: A Protoype Phase Change Memory Storage Array. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Storage and File Systems*, HotStorage'11, pages 2–2, Berkeley, CA, USA, 2011. USENIX Association.
[6] A. Awad, S. Blagodurov, and Y. Solihin. Write-Aware Management of NVM-based Memory Extensions. In *Proceedings of the 2016 International Conference on Supercomputing*, ICS '16, pages 9:1–9:12, New York, NY, USA, 2016. ACM.
[7] A. Awad, P. Manadhata, S. Haber, Y. Solihin, and W. Horne. Silent Shredder: Zero-cost Shredding for Secure Non-Volatile Main Memory Controllers. In *ACM SIGPLAN Notices*, volume 51, pages 263–276. ACM, 2016.
[8] C. R. Ferenbaugh. PENNANT: an Unstructured Mesh Mini-App for Advanced Architecture Research. *Concurrency and Computation: Practice and Experience*, 27(17):4555–4572, 2015.
[9] G. Gunow, J. Tramm, B. Forget, K. Smith, and T. He. Simplemoc-a performance abstraction for 3d moc.
[10] S. D. Hammond, A. F. Rodrigues, and G. R. Voskuilen. Multi-level memory policies: What you add is more important than what you take out. In *MEMSYS 2016*, pages 88–93, 2016.
[11] M. A. Heroux, D. W. Doerfler, P. S. Crozier, J. M. Willenbring, H. C. Edwards, A. Williams, M. Rajan, E. R. Keiter, H. K. Thornquist, and R. W. Numrich. Improving Performance via Mini-applications. Technical Report SAND2009-5574, Sandia National Laboratories, 2009.
[12] I. Karlin, J. Keasler, and R. Neely. Lulesh 2.0 updates and changes. Technical Report LLNL-TR-641973, Lawrence Livermore National Lab, August 2013.
[13] L. L. N. Lab. Hydrodynamics Challenge Problem, Lawrence Livermore National Laboratory. Technical Report LLNL-TR-490254, Lawrence Livermore National Lab, 2013.

[14] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 2–13. ACM, 2009.

[15] K. T. Malladi, B. C. Lee, F. A. Nothaft, C. Kozyrakis, K. Periyathambi, and M. Horowitz. Towards Energy-Proportional Datacenter Memory with Mobile DRAM. In *ACM SIGARCH Computer Architecture News*, volume 40, pages 37–48. IEEE Computer Society, 2012.

[16] P. J. Nair, C. Chou, B. Rajendran, and M. K. Qureshi. Reducing Read Latency of Phase Change Memory via Early Read and Turbo Read. In *High Performance Computer Architecture (HPCA), 2015 IEEE 21st International Symposium on*, pages 309–319. IEEE, 2015.

[17] P. J. Nair, D.-H. Kim, and M. K. Qureshi. ArchShield: Architectural Framework for Assisting DRAM Scaling by Tolerating High Error Rates. In *ACM SIGARCH Computer Architecture News*, volume 41, pages 72–83. ACM, 2013.

[18] M. K. Qureshi, M. M. Franceschini, and L. A. Lastras-Montano. Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–11. IEEE, 2010.

[19] 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 *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 14–23, New York, NY, USA, 2009. ACM.

[20] M. K. Qureshi, V. Srinivasan, and J. A. Rivers. Scalable High Performance Main Memory System using Phase-Change Memory Technology. *ACM SIGARCH Computer Architecture News*, 37(3):24–33, 2009.

[21] A. F. Rodrigues, K. S. Hemmert, B. W. Barrett, C. Kersey, R. Oldfield, M. Weston, R. Risen, J. Cook, P. Rosenfeld, E. CooperBalls, and B. Jacob. The Structural Simulation Toolkit. *SIGMETRICS Perform. Eval. Rev.*, 38(4):37–42, Mar. 2011.

[22] J. R. Tramm, A. R. Siegel, T. Islam, and M. Schulz. Xsbench-the development and verification of a performance abstraction for monte carlo reactor analysis.

[23] J. Wang, X. Dong, G. Sun, D. Niu, and Y. Xie. Energy-efficient Multi-level Cell Phase-Change Memory System with Data Encoding. In *Computer Design (ICCD), 2011 IEEE 29th International Conference on*, pages 175–182. IEEE, 2011.