SANDIA REPORT

SAND2017-1830
Unlimited Release
Printed February, 2017

Messier: A Detailed NVM-Based DIMM Model for the SST Simulation Framework

A. Awad, S.D. Hammond, G.R. Voskuilen, C. Hughes, A.F. Rodrigues, and R.J. Hoekstra
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM, 87185

{aawad, sdhammo, grvosku, chughes, afrodri, rjhoeks}@sandia.gov

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Sandia National Laboratories

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.osti.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/help/ordermethods.asp?loc=7-4-0#online

SAND2017-1830
Unlimited Release
Printed February, 2017

Messier: A Detailed NVM-Based DIMM Model for the SST Simulation Framework

A. Awad, S.D. Hammond, G.R. Voskuilen, C. Hughes, A.F. Rodrigues, and R.J. Hoekstra
MS 1318
Center for Computing Research
Sandia National Laboratories
Albuquerque, NM, 87185-1318

{aawad, sdhammo, grvosku, chughes, afrodri, rjhoeks}@sandia.gov

Acknowledgment

Sandia National Laboratories is a multi-mission laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

The Structural Simulation Toolkit is a high-performance, multi-node, threaded parallel discrete event simulation framework which supports the analysis of advanced computing architectures. We are grateful to the numerous funding sources who have contributed to the development of SST.

Contents

Chapter 1

Messier: A Detailed NVM-Based DIMM Model for the SST Simulation Framework

1.1 Overview of Messier

DRAM technology is the main building block of main memory, however, DRAM scaling is becoming very challenging. The main issues for DRAM scaling are the increasing error rates with each new generation, the geometric and physical constraints of scaling the capacitor part of the DRAM cells, and the high power consumption caused by the continuous need for refreshing cell values. At the same time, emerging Non-Volatile Memory (NVM) technologies, such as Phase-Change Memory (PCM), are emerging as promising replacements for DRAM. NVMs, when compared to current technologies e.g., NAND-based flash, have latencies comparable to DRAM. Additionally, NVMs are non-volatile, which eliminates the need for refresh power and enables persistent memory applications. Finally, NVMs have promising densities and the potential for multi-level cell (MLC) storage.

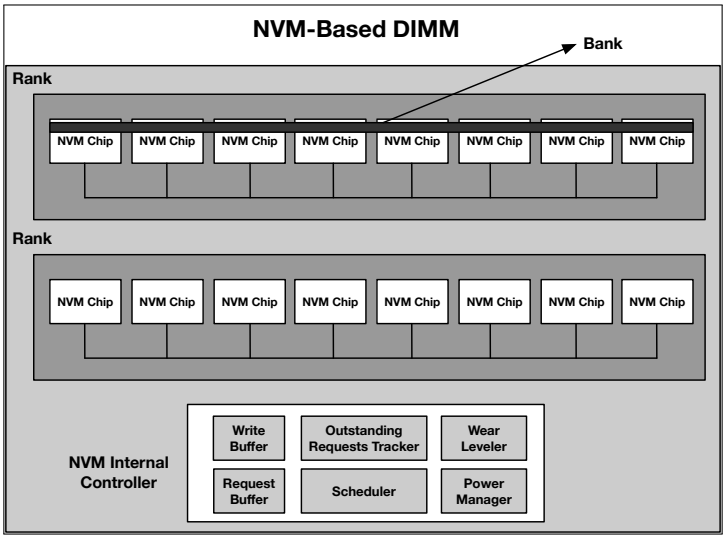

Figure 1.1. Simple illustration of the main components of an NVM-based DIMM.

Messier is proposed as an open-source academic model to study the architectural trade-offs, trends and

design options, when using NVM-based DIMMs. The key advantage of Messier its ability to model a modern NVM-based DIMM similar to what will most likely appear on the market. Accordingly, Messier enables providing architectural insights and design suggestions to reshape the next generation of NVM-based DIMMs. Messier is adopting a model similar to early prototypes of NVM-based DIMMs[2], and the designs appear on publicly available industrial patent applications[1].

Figure 1.1 depicts the overall picture of the NVM-based DIMM model that Messier adopts. The key aspects of the model are: an internal logic controller, the availability of an internal persistent write buffer to mask the write latency, DRAM-like organization of banks and ranks, and a power management unit that limits the number of current reads and writes based on a pre-defined power budget.

1.1.1 Why is Modeling Non-Volatile Memories Important?

Emerging NVMs have different characteristics compared to DRAM such as asymmetric read and write latencies and significant differences in read access latencies for different access patterns[3, 4, 5, 6]. Accordingly, projecting the performance of applications when using NVMs, and their sensitivity to different design parameters, can help us understanding how to tune the design of future systems to take advantage of emerging NVMs without sacrificing performance.

1.1.2 What Analyses can the Messier Model Support?

- **The Internal Design of NVM-based DIMMs:** Messier allows users to vary the number of ranks, the number of banks, the size of row buffers, the maximum number of outstanding requests, the write buffer size, the write buffer flushing policy, the maximum number of concurrent reads/writes based on power budget, and the memory interleaving policy.
- **The Latencies of Emerging NVMs:** Messier allows users to vary the latencies of the modeled NVM devices, allowing researchers to project the performance of different NVM technologies, e.g., Phase-Change Memory, Memristor or Spin-Transfer Torque. Messier also distinguishes between read and write latencies, hence enabling NVMs with asymmetric read/write latencies.

The details of how to use Messier and vary its parameters will be discussed in the next sections.

Chapter 2

Messier Specification (Version 1.0)

In this chapter, we detail the specifications of the Messier SST element and its model. Messier is implemented as an element of the Structural Simulation Toolkit (SST) simulator [7]. Figure 2.1 depicts the architecture of Messier, including the main classes:

- **NVM_DIMM Class**: The TLB class is defined inside *NVM_DIMM.h* and implemented in *NVM_DIMM.cc*. The NVM_DIMM class is used to define NVM-Based DIMM structures with the specified parameters. Each NVM_DIMM object models a single DIMM with its own internal ranks and banks.
- **RANK Class**: The RANK class is defined inside *Rank.h*. The RANK class is used to define a single rank of NVM chips.
- **BANK Class**: The BANK class is defined inside *Bank.h*. It defines the bank structure inside a rank.
- **NVM_WRITE_BUFFER Class**: The WriteBuffer class is defined inside *Bank.h*. It defines the bank structure inside a rank.
- **Messier Class**: The Messier class is defined inside *Messier.h* and implemented in *Messier.cc* and *libMessier.cpp*. The Messier class is mainly used to instantiate an NVM_DIMM objects as SST components.

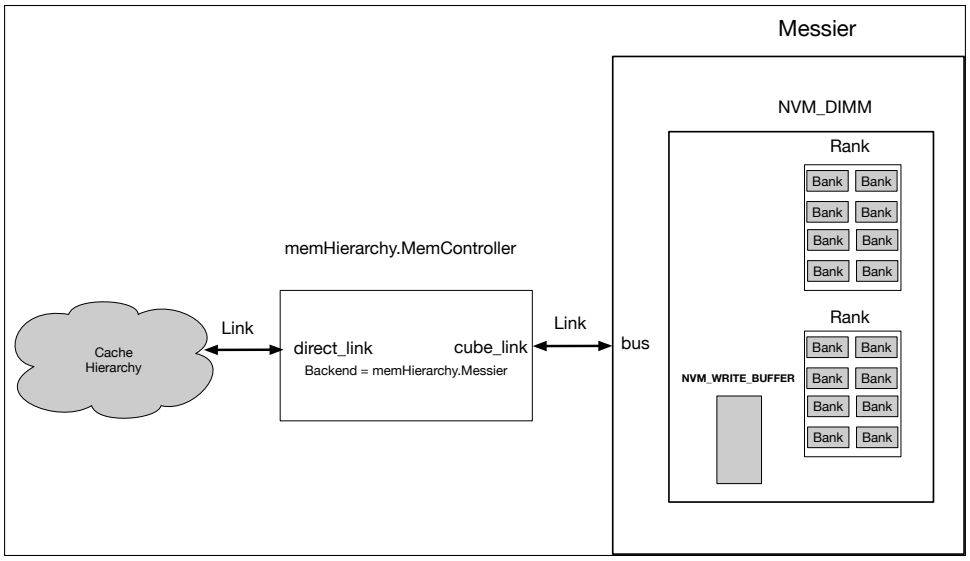

Figure 2.1. The architecture of Messier.

The key object inside the Messier component is NVM-DIMM. NVM-DIMM contains severals structures, shown in Figure 2.2. At each clock cycle, the NVM-DIMM controller does the following: (1) check if the write buffer has more entries than the specified threshold, i.e., *write priority inversion*, or no current transactions, then dispatch the write to the corresponding bank. Note that this process allocates the rank for the time needed for sending the data and the command, while the bank is allocated according to the specified latency of a write to the PCM chips. (2) the transaction queue is checked to dispatch a request, based on the scheduling policy, e.g., prioritize row buffer hits then old requests (3) a write request is placed on the write buffer, while a read request is sent to the corresponding bank and placed in the outstanding requests queue. Note that before dispatching any request, we need to check that both the corresponding banks and ranks are idle, in addition to not exceeding the maximum number of permitted outstanding requests (reads or writes).

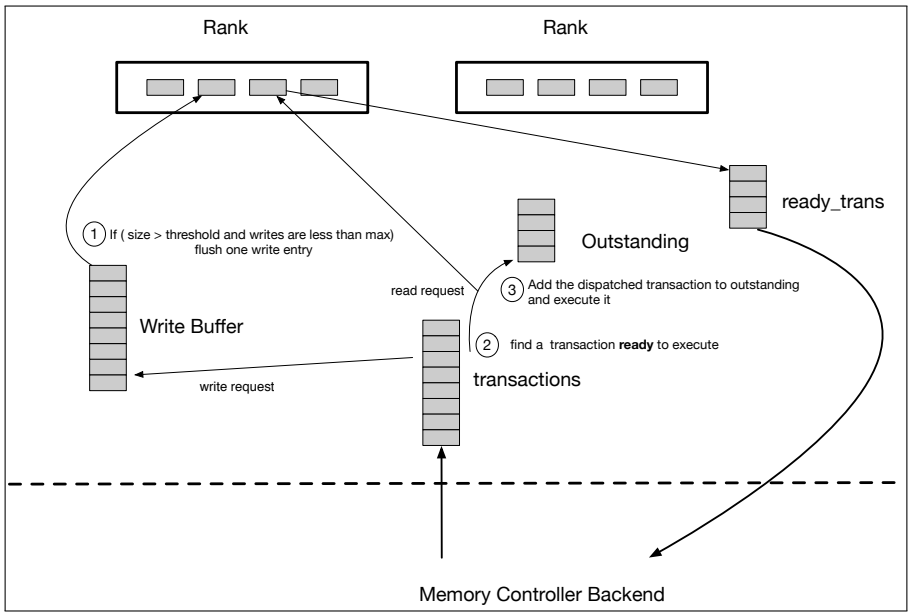

Figure 2.2. The algorithm and internal structures of NVM_DIMM.

In the next section, we will study the details of each class and its parameters.

2.1 Example of Messier Usage

```
# Instantiating an instance of the Messier
messier_inst = sst.Component("NVMmemory", "Messier")
# Note that the size is indicated at the memory backend

messier_inst.addParams({
      "tCL" : "70", # This indicates the latency (cycles) of reading data from the row buffer
      "tRCD" : "200", # Indicates the latency (cycles) installing a row from the NVM chips into the
          row buffer
      "clock" : "1GHz", # The frequency operating the DIMM controller
      "tCL_W" : "100", # This is the latency (cycles) of writing a row in the NVM chips
      "write_buffer_size" : "32", # The max number of entries in the write buffer
```

```
    "flush_th" : "90", # The threshold of filled entries (here it is 90\% of max entries) when
        writes are given priority to be flushed from the write buffer
    "num_banks" : "16", # this is the number of banks inside each rank
    "max_outstanding" : "32", # this indicates the maximum number of outstanding reads allowed
    "max_current_weight" : "160", # this indicates the maximum relative power allowed
    "read_weight" : "5", # this indicates the relative power of each currently executing read
        request
    "write_weight" : "50", # this indicates the relative power of each currently executing write
        request
    "max_writes": "8", # this indicates the maximum number of concurrent writes to NVM chips
    "row_buffer_size" : "8192", # this indicates the row buffer size in bytes
    "cacheline_interleaving": "0", # 0 means row interleaving (consecutive cachelines go to the
        same bank), while 1 means cacheline interleaving (consecutive cachelines go to
        consecutive banks)
})
```

In the next section, we will study an example of the performance sensitivity to Messier's parameters.

Chapter 3

Application Performance Sensitivity to Messier Parameterization

In this chapter, we will vary different parameters for the Messier unit and study how this affects the performance.

3.1 Methodology

We run our simulation using the SST simulator [7] modified to include the Messier model. We use the XSBench [8], MiniFE and Pennant miniapps as case studies for our demo. In each run, we simulate 8 cores with 2GHz frequency, while running 8 threads from each miniapp until at least one core executes 20M instructions (approximately 160M total instructions).

For each cores, we model an issue width of 3 instructions and assume a maximum of 16 outstanding memory requests. Our default configurations for Messier is shown below:

```
messier_params = {
      "clock" : clock, # same as the cpu clock
      "tCL" : 30,
      "tRCD" : 300,
      "max_writes" : 4,
      "tCL_W" : 1000,
      "write_buffer_size" : 32,
      "flush_th" : 90,
      "num_banks" : 32,
      "max_outstanding" : 32,
      "max_current_weight" : 32*50,
      "read_weight" : "5",
      "write_weight" : "5",
      "row_buffer_size" : "8192",
      "cacheline_interleaving" : "1",
}
```

3.1.1 NVM Read Latency

The read latency of NVM devices is one of the most critical aspects, mainly due to the nature of read requests being in the critical path. NVM read latency can vary based on the technology, number of levels in the cell, etc. Applications that exhibit good spatial behavior will enjoy a higher row buffer hit rate, avoiding accessing NVM chips for a large percentage of accesses.

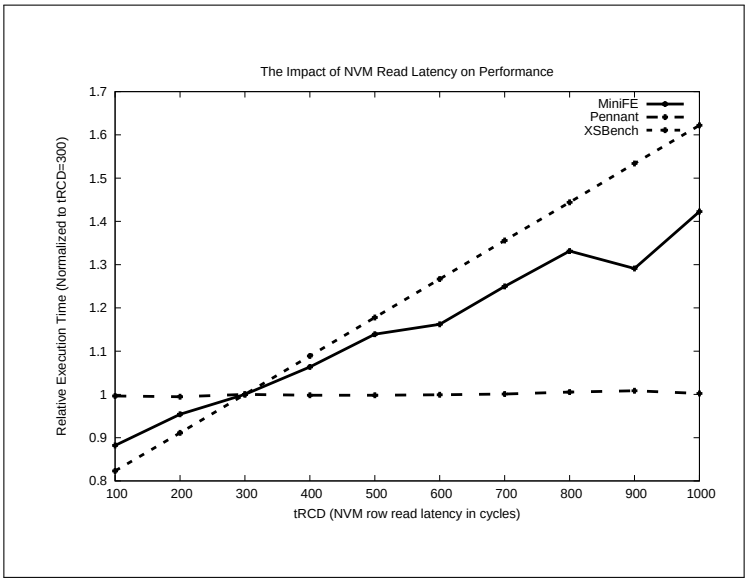

Figure 3.1. The impact of NVM read latency on performance.

As shown in Figure 3.1, applications with poor spatial locality, e.g., XSBench, are very sensitive to the NVM latency, as most of the accesses will go directly to the NVM chips, due to poor row buffer locality. In contrast, Pennant, which exhibits a better low buffer locality, is only slightly impacted by the NVM latency.

3.1.2 The Impact of Maximum Outstanding Read Requests

The maximum number of concurrent read requests that can be serviced in the NVM-DIMM affects the performance significantly. The impact of this parameter depends on how memory intensive the application is (the number of memory requests that reach the memory per time unit) and the number of bank conflicts caused by the access pattern.

Figure 3.2 shows the impact of this parameter on the performance of different applications. We can observe that XSBench benefits the most from increasing the number of outstanding requests, mainly due to its memory intensity and the randomness of the memory requests (less accesses to the same bank). MiniFE also shows some sensitivity to the maximum number of outstanding requests. In contrast, Pennant rarely benefits from increasing the maximum number of allowed outstanding requests.

3.1.3 The Impact of Write Latency on Performance

The write latency of NVMs can vary significantly between different technologies. While the write buffer can help mask the write latency, a high write latency can quickly fill up the write buffer and place back-pressure on the memory controller, which reduces the number of requests that can be submitted to the NVM-DIMM. Figure 3.3 shows the impact of the write latency on the three test applications.

We observe that XSBench has less sensitivity to the write latency, which is likely due to the small number of dirty cache blocks compared to other applications. In contrast, we observe that Pennant is highly sensitive to the write latency. Note that our write policy is based on a threshold value, where if the percentage of write entries exceeds that threshold, the writes become a higher priority than reads. Additionally, the controller

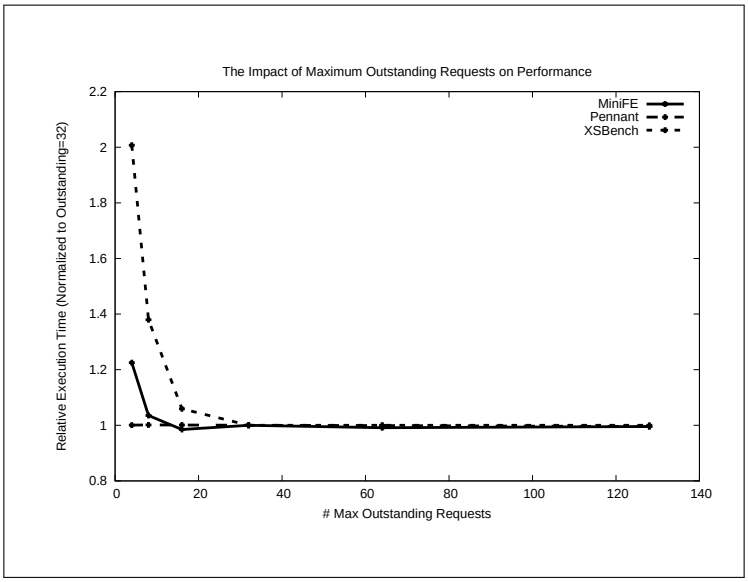

Figure 3.2. The impact of the maximum concurrent read requests on performance.

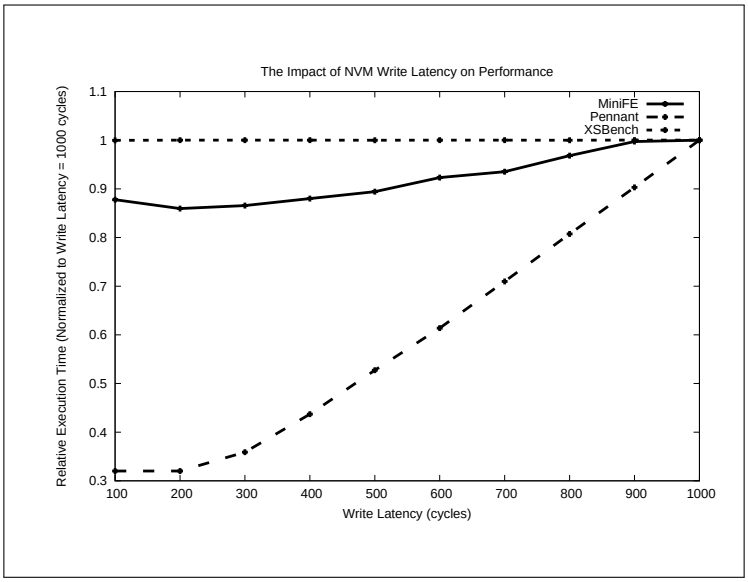

Figure 3.3. The impact of the write latency on performance.

flushes write requests from the write buffer whenever the transaction queue is empty.

3.1.4 The Impact of Maximum Number of Concurrent Writes on Performance

While this parameter can be restricted by the power budget, allowing a high number of concurrent writes enables fast spilling of writes from the write buffer, reducing the possibility of back-pressuring the memory controller. Figure 3.4 shows the sensitivity of the maximum concurrent writes on the performance of the tested applications.

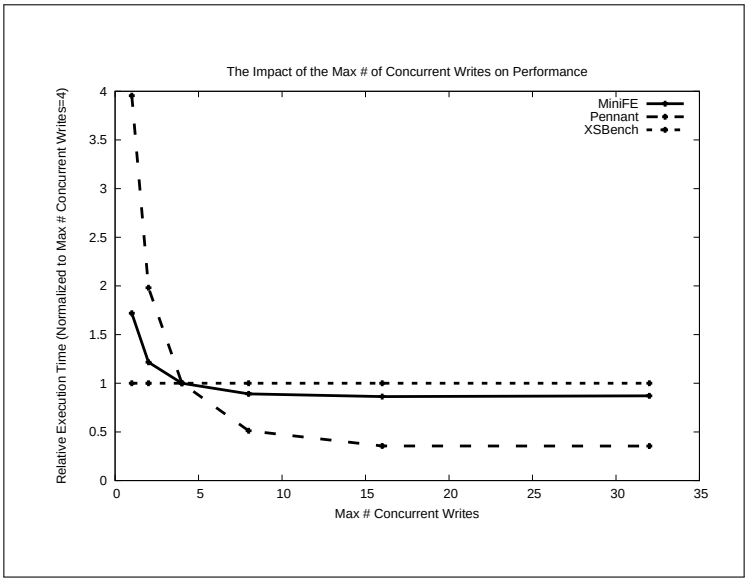

Figure 3.4. The impact of the maximum number of outstanding writes on performance.

From the figure, we can observe that Pennant and MiniFE are very sensitive to this parameter. Meanwhile, XSBench is barely affected, which is consistent with the write latency study in Section 3.1.3.

3.1.5 The Impact of the Number Banks on Performance

The number of banks determines the level of potential parallelism inside each rank. Figure 3.5 shows how the applications are affected by the number of banks parameter.

We can observe that most applications benefit from increasing the number of banks up to 32, however, the performance gains start leveling-off at 32 banks, because the memory controller (the backend), can only handle a maximum of 32 pending requests to the NVM-DIMM.

3.1.6 The Impact of Interleaving

As mentioned earlier, Messier enables using cacheline interleaving (default), where consecutive cachelines go to consecutive banks, and row interleaving, where consecutive cachelines go to the same bank. The three

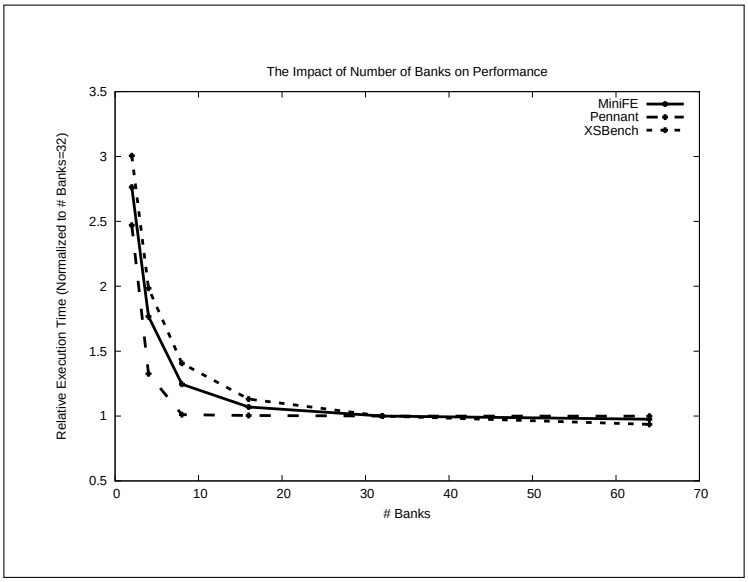

Figure 3.5. The impact of the number of banks on performance.

graphs in Figure 3.6 show the impact of the interleaving on each application while varying the maximum number of outstanding requests.

From these results, we can observe that all three applications receive some benefit when using row interleaving over cacheline interleaving.

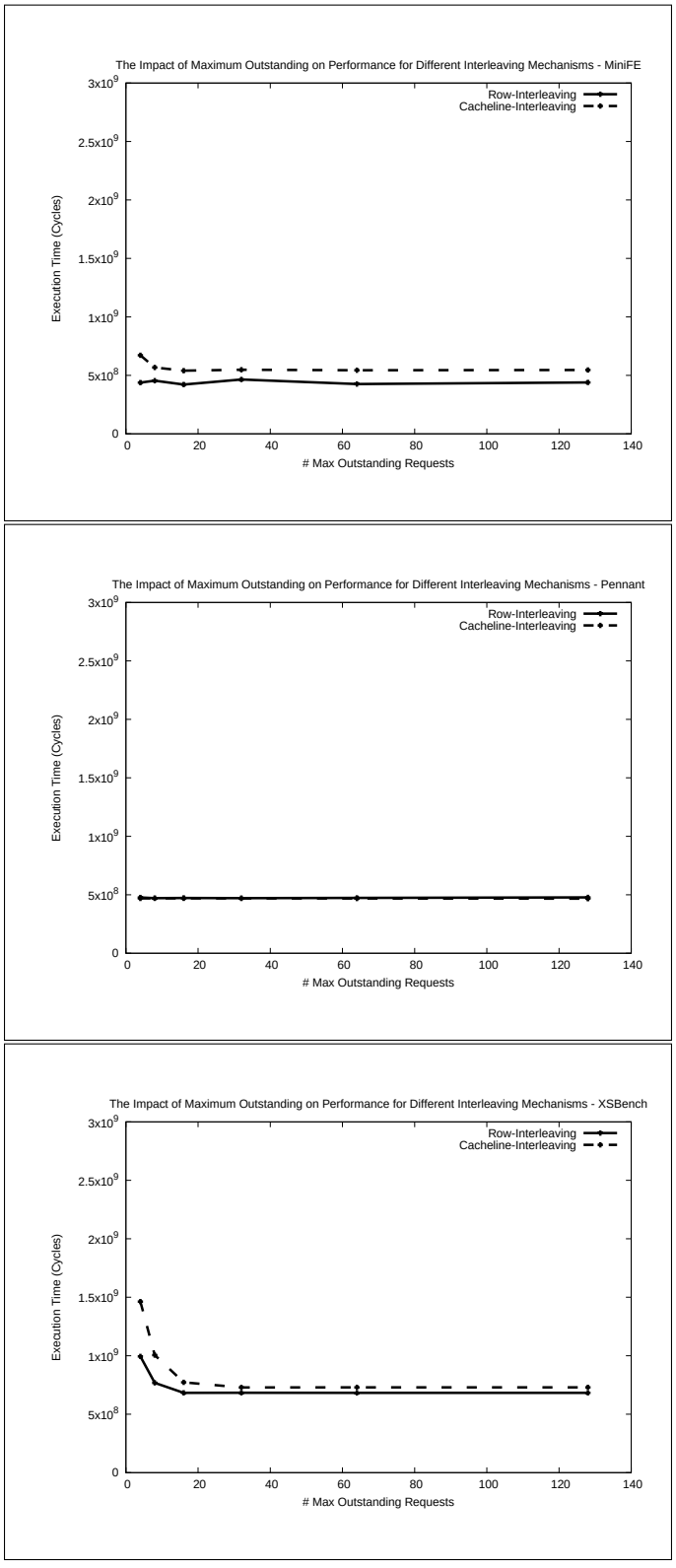

Figure 3.6. The impact of the interleaving policy on the performance.

Chapter 4

Conclusion

In conclusion, we have presented Messier as a detailed NVM-based DIMM model for the SST simulation framework. We explained the options and parameters available in the model and discussed how each can be configured and how the performance would be affected. Finally, we evaluated MiniFe, Pennant, and XSBench as a short case study to show how application performance can vary with differing configurations of Messier's components. We hope that Messier will be used to evaluate the impact of using non-volatile memories on future system for different workloads.

References

[1] Published U.S Patent Application. Dynamic partial power down of memory-side cache in a 2-level memory hierarchy, PCT/US2011/066302. URL https://www.google.com/patents/US20140304475.

[2] Ameen Akel, Adrian M. Caulfield, Todor I. Mollov, Rajesh K. Gupta, and Steven 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. URL http://dl.acm.org/citation.cfm?id=2002218.2002220.

[3] Amro Awad, Sergey Blagodurov, and Yan 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. ISBN 978-1-4503-4361-9. doi: 10.1145/2925426.2926284. URL http://doi.acm.org/10.1145/2925426.2926284.

[4] Amro Awad, Pratyusa Manadhata, Stuart Haber, Yan Solihin, and William Horne. Silent shredder: Zero-cost shredding for secure non-volatile main memory controllers. *SIGOPS Oper. Syst. Rev.*, 50(2): 263–276, March 2016. ISSN 0163-5980. doi: 10.1145/2954680.2872377. URL http://doi.acm.org/10.1145/2954680.2872377.

[5] Benjamin C Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting phase change memory as a scalable dram alternative. In *ACM SIGARCH Computer Architecture News*, volume 37, pages 2–13. ACM, 2009.

[6] Moinuddin K Qureshi, Michele M Franceschini, Ashish Jagmohan, and Luis A Lastras. Preset: improving performance of phase change memories by exploiting asymmetry in write times. *ACM SIGARCH Computer Architecture News*, 40(3):380–391, 2012.

[7] Arun F Rodrigues, K Scott Hemmert, Brian W Barrett, Chad Kersey, Ron Oldfield, Marlo Weston, R Risen, Jeanine Cook, Paul Rosenfeld, E CooperBalls, et al. The Structural Simulation Toolkit. *ACM SIGMETRICS Performance Evaluation Review*, 38(4):37–42, 2011.

[8] John R Tramm, Andrew R Siegel, Tanzima Islam, and Martin Schulz. XSBench - The Development and Verification of a Performance Abstraction for Monte Carlo Reactor Analysis. In *PHYSOR 2014 - The Role of Reactor Physics toward a Sustainable Future*, Kyoto.

DISTRIBUTION:

1 MS 0899 Technical Library, 9536 (electronic copy)

Sandia National Laboratories