Exploring Use-cases for Non-Volatile Memories in support of HPC Resilience

Onkar Patil
Dept. of Computer Science, North Carolina State University
Raleigh, North Carolina
opatil@ncsu.edu

Saurabh Hukerikar
Computer Science Research, Oak Ridge National Laboratory
Oak Ridge, Tennessee
hukerikarsr@ornl.gov

Frank Mueller
Dept. of Computer Science, North Carolina State University
Raleigh, North Carolina
mueller@cs.ncsu.edu

Christian Englemann
Computer Science Research, Oak Ridge National Laboratory
Oak Ridge, Tennessee
engelmannc@ornl.gov

CCS CONCEPTS

• **Computer Architecture** → **High Performance Computing**; Memory Architecture; • **Software Engineering** → *Resilience*; • **Memory Devices** → Non-Volatile Memory;

KEYWORDS

Resilience, HPC, NVM, Persistence, Checkpointing

ACM Reference format:
Onkar Patil, Saurabh Hukerikar, Frank Mueller, and Christian Englemann. 2017. Exploring Use-cases for Non-Volatile Memories in support of HPC Resilience. In *Proceedings of ACM Woodstock conference, Denver, CO, USA, November 2017 (SC'17),* 2 pages.
DOI: 10.475/123_4

1 INTRODUCTION

The exascale supercomputer will be different than its predecessors in many ways. Exaflop computation capabilities will be realized by a vast ensemble of processors, co-processors, accelerators and memory devices all connected over different forms of high bandwidth interconnects. These future systems need to be more resilient along with maintaining a balance between performance and power consumption and minimizing their trade-offs.[1][2] The emergence of non-volatile memory (NVM) technologies, such as phase change memory (PCM) will enable memory devices that can maintain state of computation in the primary memory architecture and will be a part of the future computing architectures. We see more potential in using these memory devices as specialized hardware rather than commodity hardware. This will lead to sophisticated methods of capturing system/application state at checkpoints and eventually increasing the consistency of the data. Utilizing persistent memory regions to improve HPC resiliency is the key aspect of this project. The unique aspect of NVM is that it retains data while being byte-addressable, i.e., it can be mapped directly in the program address space.[4][5] This feature opens new avenues for HPC Resilience methodologies where lightweight and efficient checkpointing and logging mechanisms can be developed. These methods can enable system and application resilience at a much finer grain and provide a required basis for programming. There is a lack of convenient programming interfaces that helps integrate these methods into the existing applications too.[3] Our aim is to develop novel resilience methodologies for future HPC systems with easy programming interfaces to tap the complete potential of persistent memory regions.

2 DESIGN

We introduce three modes of usage of NVM devices to aid in ensuring HPC resilience.

- **NVM-based Main Memory** : This mode provides a unified memory region that treats NVM and DRAM uniformly. The memory allocated on NVM is crash consistent. Critical data structures can be directly allocated on persistent memory devices and can be accessed directly by the processor.
- **Application-directed Checkpointing** : In this mode, the API allows users to select which data needs to be protected through persistence to improve efficiency and avoid space and time overheads. We maintain a copy of the data structure in persistent memory while the application computes on the data structure in DRAM.
- **Data versioning** : This mode preserves multiple snapshots of the application state such that the application recovery process may select among previous versions. It requires maintenance of data values, structure, and metadata related for each version. Along with a persistent copy we also store another copy of the same data structure in the persistent memory region transparently.

The design strategy behind these modes of persistent memory usage is to enable checkpointing at the data structure level. We backup some data structures that are more critical than others at different stages of the application in terms of failure recovery. We have developed a simple API to prototype the above three modes of memory usage. The API has functions that help with the initializing stage, allocation stage, updating stage and finalizing stage.

3 EXPERIMENTAL EVALUATION

We evaluate our implementation on a 16-node cluster with Dual socket, Quad-Core AMD Opteron, 128 GB DRAM memory, Intel SSD varying in size from 100GB to 256GB. We test our implementation against the DGEMM benchmark[1]. We tested for 4, 8 and 16-node configurations for a matrix sizes of 1000, 2000 and 3000 elements. We collected execution times and GFLOPS averaged them over 100 runs for every configuration. We observe that DGEMM performs similarly when used both with only DRAM memory and only Persistent memory. Its performance is also similar between Application directed checkpointing and Data Versioning. Using only one kind of memory shows around 2 orders of magnitude better performance than using persistent memory as a backup of DRAM memory. This difference in performance is due to a very naive mapping and lookup algorithm used in our implementation with O(n) complexity. The node scaling does not affect the performance at all. The performance deteriorates quickly as the size increases for the Application directed checkpointing and Data Versioning usage modes which is depicted by the rapid decrease in GFLOPS. The performance degrades partly due to the inefficient lookup mechanism and increase in cache misses because of increased problem size.

4 FUTURE WORK

In the future, we would further develop the memory usage modes to make them more efficient and maintain complete system state with minimal overhead and support more complex applications. Here, we developed lightweight recovery mechanisms to work with the checkpointing schemes to reduce downtime and rollback time. We would like to take this idea further and combine them with intelligent policies that can help build resilient static and dynamic runtime system.

5 CONCLUSION

In this work, we showed that Non-volatile memory devices can be used as specialized hardware for improving the resilience of the applications and we demonstrated three potential memory usage models with consistent performance for node and problem size scaling.

REFERENCES

[1] John Daly, Bill Harrod, Thuc Hoang, Lucy Nowell, Bob Adolf, Shekhar Borkar, Nathan DeBardeleben, Mootaz Elnozahy, Mike Heroux, David Rogers, Rob Ross, Vivek Sarkar, Martin Schulz, Marc Snir, Paul Woodward, Rob Aulwes, Marti Bancroft, Greg Bronevetsky, Bill Carlson, Al Geist, Mary Hall, Jeff Hollingsworth, Bob Lucas, Andrew Lumsdaine, Tina Macaluso, Dan Quinlan, Sonia Sachs, John Shalf, Tom Smith, Jon Stearley, Bert Still, and Jon Wu. 2012. *Report: Inter-Agency Workshop on HPC Resilience at Extreme Scale.* Technical Report. Advanced Computing Systems, National Security Agency.

[2] Saurabh Hukerikar and Christian Engelmann. 2016. *Resilience Design Patters: A Structured Approach to Resilience at Extreme Scale, version 1.1.* Technical Report. Oak Ridge National Laboratory.

[3] Arash Rezaei. 2016. *Fault Resilience for Next Generation HPC Systems.* North Carolina State University.

[4] Daniel Wong, GS Lloyd, and MB Gokhale. 2013. *A memory-mapped approach to checkpointing.* Technical Report. Lawrence Livermore National Laboratory (LLNL), Livermore, CA.

[5] Michael Wu and Willy Zwaenepoel. 1994. eNVy: a non-volatile, main memory storage system. In *ACM SIGOPS Operating Systems Review*, Vol. 28. ACM, 86–97.

[1] This research used resources of the Oak Ridge Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC05-00OR22725.