Large-Scale Adaptive Mesh Simulations Through Non-Volatile Byte-Addressable Memory

Bao Nguyen, Hua Tan, Xuechen Zhang
School of Engineering and Computer Science
Washington State University
{bao.nguyen,hua.tan,xuechen.zhang}@wsu.edu

Kei Davis
Applied Computer Science Group
Los Alamos National Laboratory
kei@lanl.gov

ABSTRACT

This paper presents a novel data structure **P**ersistent **M**erged octree (PM-octree) for both meshing and in-memory storage of persistent octrees using NVBM. It is a multi-version data structure and can recover from failures using its earlier persistent version stored in NVBM. In addition, we have designed a feature-directed sampling approach to help dynamically transform the PM-octree layout for reducing NVBM-induced memory write latency. Our results show that simulations implemented using PM-octree have good scalability and provide consistency upon failure.

1 INTRODUCTION

Octree-based adaptive mesh refinement on high-performance computing (HPC) clusters has enabled simulation of complex physical phenomena. Computational scientists seek to run their physics models at ever-larger length- and time-scales such that memory demands for running such simulation is significant even on supercomputers [3]. At the same time, it has become increasingly difficult to scale DRAM. Therefore, we must seek more cost-effective solutions to extend memory capacity available to large-scale meshing and visualization using octree data structures. The *non-volatile, byte-addressable memory* (NVBM) technologies, e.g., Phase Change Memory (PCM) [2], STT-MRAM [1], and Memristor [8], are promising enabling technologies for alleviating these problems.

Three challenges exist in the design of an NVBM-aware octree because some physical aspects (e.g., write latency and endurance) of NVBM are so much different from those of DRAM. First, NVBM writes may incur high latency while octree meshing operations can be write-intensive. Therefore, placing the octrees in NVBM may expose the long write latency and result in significant loss of performance. Second, the existing octree data structure is not durable for NVBM. Corruption is likely when hardware or software failures happen. Last but not least, the legacy octree solutions do not handle special pointers linking persistent octants in NVBM and volatile octants in DRAM. In addition, we need a library that can automatically handle such special pointers during failure recovery without introducing extra complexity for application developers.

In this paper, we propose a novel data structure *persistent merged octree* (PM-octree) [1] for both meshing and in-memory storage of persistent version of octrees. It is a multi-version data structure and no special instructions for enforcing ordering of memory access is needed because our algorithms can guarantee at least one version of the octree is consistent while updating its newer version. If a failure happens, the consistent version of the octree will be accessed for restarting a program. PM-octree is partitioned so that NVBM-induced additional memory latencies can be effectively shielded. We propose a feature-directed sampling approach to identify popular sub-domains using application-level knowledge about data features. In addition, we also dynamically adjust the size of subtrees in DRAM and NVBM to improve memory efficiency according to the memory utilization tracked by operating systems.

[1] The work has been published in SC'17 [4].

2 DESIGN OF PM-OCTREE

The design objective of PM-octree is to effectively utilize NVBM for memory extension and failure recovery. This is achieved by introducing a ***p**ersistent **m**erged octree* (PM-octree) for meshing and solving using multi-versions of octrees, as shown in Figure 1. PM-octree is a persistent octree data structure, which keeps two of

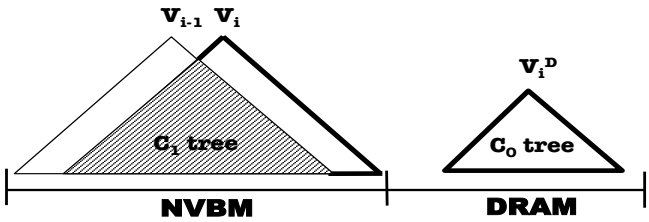

Figure 1: An illustration of PM-octree. V_{i-1} and V_i: root nodes of the octrees operated on time steps $i-1$ and i. While V_{i-1} is entirely stored in NVBM, V_i is partitioned into two segments according to data access locality. The C_0 tree composes of frequently accessed subtrees and is stored in DRAM. The C_1 tree composes of subtrees which are less frequently accessed and is stored in NVBM. V_{i-1} and V_i share octants (in the shaded area). Only V_i is visible to applications during a normal execution. V_{i-1} is used for restarting applications upon failures.

its latest versions V_{i-1} and V_i. V_{i-1} is a persistent version of the octree saved at the end of the simulation of the time step T_{i-1} and resides entirely in NVBM leveraging its non-volatility for failure recovery. V_i is a version of the octree being actively accessed at the time step T_i. For V_i, its frequently accessed subtrees (C_0) are stored in DRAM and the rest of the tree (C_1) resides in NVBM. PM-octree supports shared octants between V_{i-1} and V_i to reduce redundant octants stored in memory, therefore, reducing memory footprint and improving memory utilization.

PM-octree has four major operations: insertion, update, deletion, and merging. An octant can be inserted into either the C_0 tree or C_1 tree, determined by its locational code. The insertion of an octant into C_0 does not incur long write latency but if an octant is inserted into

C_1 in NVBM, multiple copying steps might be needed to propagate updates towards the root. For updating operations, only the octants of C_0 and the octants of C_1 but not shared with V_{i-1} can be updated in place. To update an octant shared by V_i and V_{i-1}, we need to first create a copy of the octant and then update it. For deletion, an octant cannot be removed directly from V_{i-1}, a copy of its parent octant will be created with an empty pointer to replace the pointer which linked to the removed octant. We can directly delete an octant in C_0 as it resides in DRAM but if an octant needs to be removed from C_1 in NVBM, we will only mark the octant as "deleted". The real deletion is handled by garbage collection (GC). V_{i-1} is used to restart an application and V_i is accessed during its normal execution. The merging routine is executed in two scenarios: (1) The size of C_0 is large so that remaining DRAM space is smaller than $threshold_{DRAM}$. To maintain efficiency, a subtree of C_0 is trimmed and merged out with C_1; (2) Simulation of time step i is completed and a persistent point of the octree needs to be saved.

Access to data in NVBM may incur high read/write latencies so dynamic transformation is designed to achieve optimal performance. We use DRAM to store frequently accessed (hot) subtrees of PM-octree and use NVBM to store less frequently accessed (cold) ones. We propose a *feature-directed sampling technique* to alter the layout of PM-octree and determine when a transformation should be executed. A transformation should be executed only when the access frequency of subtree i in NVBM ($Freq_i^{NVBM}$) is significantly larger than the access frequency of subtree j in DRAM ($Freq_j^{DRAM}$). We use the following steps to predict the access frequency. (1) Randomly select N_{sample} octants in a subtree; (2) Pre-execute feature functions; (3) The access frequency of the subtree is computed as the total number of 1s returned by the feature functions; (4) Compute the ratio of access using access frequency and check whether it is larger than a threshold $T_{transform}$. If it is true, then it indicates that the data access pattern shall be changed and it is necessary to re-layout PM-octree.

3 EVALUATION

To test our idea we model NVBM using DRAM on real computer servers. We use an emulation based approach and our NVBM emulator introduces extra latency for NVBM writes and reads. We create delays using a software spin loop [5, 7]. To evaluate the correctness and scalability of PM-octree based adaptive meshing algorithms, we develop a program to simulate *droplet ejection*, as the driving scientific workload. The simulation of the droplet ejection is implemented using three octree implementations. (1) *In-core-octree*: the existing octree data structure used in the meshing algorithm in Gerris. During its simulation all octants are stored in DRAM.(2) *Out-of-core-octree*: it is a data structure used in the Etree library [6] for meshing using slow storage devices, e.g., hard disks. (3) *PM-octree*: the data structure described in Section 2.

We first study weak scaling of the simulation. We measure their execution times while increasing the problem size from 1.2M to 1077M elements and the number of processors from 1 to 1000. PM-octree achieves similar weak scaling as in-core-octree. The times on refining and balancing increase as a logarithm of problem size suggests that PM-octree has good scalability.

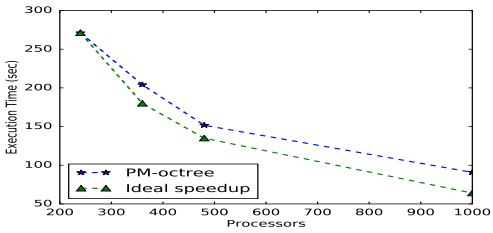

Figure 2: Execution time of the simulation with PM-octree, compared to that with ideal speedup.

Second, we study strong scaling of the simulation with PM-octree. We compute the speedup when the problem size is kept constant and the number of processors is increased. The execution time speedup with PM-octree is close to the ideal speedup on 240-1000 processors, as shown in Figure 2.

4 CONCLUSIONS

We have presented the design and implementation of PM-octree data structure for large-scale simulations using octrees. It not only can effectively extend memory capacity using NVBM but also support near-instantaneous failure recovery. It supports orthogonal persistence for applications and provides an easy-to-program interface. A real-world flow simulation program is developed using PM-octree to simulate droplet ejection in inkjet printing. Our experimental results show that the simulations implemented using PM-octree have good scalability up to 1.1 billion elements on 1000 processors on the Titan supercomputer.

REFERENCES

[1] Dmytro Apalkov, Alexey Khvalkovskiy, Steven Watts, Vladimir Nikitin, Xueti Tang, Daniel Lottis, Kiseok Moon, Xiao Luo, Eugene Chen, Adrian Ong, Alexander Driskill-Smith, and Mohamad Krounbi. 2013. Spin-transfer Torque Magnetic Random Access Memory (STT-MRAM). *J. Emerg. Technol. Comput. Syst.* 9, 2 (May 2013).

[2] Geoffrey W Burr, Matthew J Breitwisch, Michele Franceschini, Davide Garetto, Kailash Gopalakrishnan, Bryan Jackson, Bülent Kurdi, Chung Lam, Luis A Lastras, Alvaro Padilla, and others. 2010. Phase change memory technology. *Journal of Vacuum Science & Technology B* 28, 2 (2010), 223–262.

[3] Tong Jin, Fan Zhang, Qian Sun, Hoang Bui, Manish Parashar, Hongfeng Yu, Scott Klasky, Norbert Podhorszki, and Hasan Abbasi. 2013. Using Cross-layer Adaptations for Dynamic Data Management in Large Scale Coupled Scientific Workflows. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC '13)*.

[4] Bao Nguyen, Hua Tan, and Xuechen Zhang. 2017. Large-scale Adaptive Mesh Simulations Through Non-volatile Byte-addressable Memory. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. ACM, New York, NY, USA, Article 27, 12 pages. DOI: https://doi.org/10.1145/3126908.3126944

[5] Jiaxin Ou, Jiwu Shu, and Youyou Lu. 2016. A High Performance File System for Non-volatile Main Memory. In *Proceedings of the Eleventh European Conference on Computer Systems (EuroSys '16)*.

[6] T. Tu, J. Lopez, and D. O'Hallaron. 2003. *The Etree Library: A System for Manipulating Large Octrees on Disk.* Technical Report CMU-CS-03-174. Carnegie Mellon School of Computer Science.

[7] Haris Volos, Sanketh Nalli, Sankarlingam Panneerselvam, Venkatanathan Varadarajan, Prashant Saxena, and Michael M. Swift. 2014. Aerie: Flexible File-system Interfaces to Storage-class Memory. In *Proceedings of the Ninth European Conference on Computer Systems (EuroSys '14)*.

[8] J. Joshua Yang and R. Stanley Williams. 2013. Memristive Devices in Computing System: Promises and Challenges. *J. Emerg. Technol. Comput. Syst.* 9, 2 (May 2013).