pmmeter: A Microbenchmark for Understanding Synchronization Cost on Persistent Memory

Hirotaka YOSHIOKA
The University of Tokyo
Tokyo, Japan
hyoshiok@tkl.iis.u-tokyo.ac.jp

Yuto HAYAMIZU
The University of Tokyo
Tokyo, Japan
haya@tkl.iis.u-tokyo.ac.jp

Kazuo GODA
The University of Tokyo
Tokyo, Japan
kgoda@tkl.iis.u-tokyo.ac.jp

Masaru KITSUREGAWA
The University of Tokyo
Tokyo, Japan
kitsure@tkl.iis.u-tokyo.ac.jp

***Abstract*—Persistent memory is an emerging memory device, which offers durable, relatively large memory space cost-effectively. This technology potentially enables a new horizon for a broad spectrum of data-intensive applications. Historically, memory space has been deemed to be volatile for application programs. Only very recently the processor industry has introduced a variety of memory synchronization instructions to ensure the durability of persistent memory. Yet, no known programming practice yields how application programs can fully exploit those instructions. This paper proposes pmmeter, a new microbenchmarking tool that we have developed to measure the primary performance of persistent memory. pmmeter allows for clarifying the performance impact that the choice of synchronization instructions incurs on a given sequence of memory access. This paper presents our experiment to demonstrate that pmmeter unveils the synchronization cost of Intel Optane DCPMM.**

***Index Terms*—Persistent memory, performance measurement, microbenchmarking tool**

I. Introduction

Persistent memory (PMEM) is an emerging memory device, which offers durable, relatively large memory space cost-effectively. Traditionally, the data memory and storage space in computer systems have been provided by two different technologies: memory technology such as dynamic random access memory (DRAM) providing volatile, relatively small, fast and costly space, and storage technology such as hard disks and NAND flash memory providing persistent, relatively large, slow and cheap space. PMEM can be positioned as an in-between technology and potentially enables a new horizon for a broad spectrum of data-intensive applications. Recent PMEM such as Intel Optane DCPMM [1] can be directly connected to memory bus, and it can work if it were the main memory device. This solution has the benefit of being able to keep backward compatibility; existing applications can transparently cost-effectively use large memory capacity, since PMEM offers a much larger capacity than DRAM.

On another front, effectively and efficiently utilizing the durability of PMEM for applications is nontrivial, even though PMEM potentially offers unique opportunities for a broad spectrum of data-intensive applications such as database systems and file systems [2]–[5]. Historically, memory space has been deemed to be volatile for application programs. Only very recently, the processor industry has introduced a variety of memory synchronization instructions to ensure the durability of PMEM. Yet, no known programming practice yields how application programs can fully exploit those instructions. Exploring and establishing effective practices is a grand technical challenge for computer systems.

This paper proposes *pmmeter*, a new microbenchmarking tool that we have developed to measure the primary performance of PMEM. pmmeter allows for clarifying the performance impact that the choice of synchronization instructions incurs on a given sequence of memory access. Our present study has experimentally tested Intel Optane DCPMM, since this is the only available commercial product in the market. Intel Optane DCPMM is the earliest commercial device of persistent memory backed by the 3D XPoint technology [6]. Unfortunately, this business has not been successful so far; Intel recently announced the decision to discontinue this product business in August [7]. Shortly after, Kioxia announced the launch of new-generation XL-FLASH devices as yet another persistent memory technology [8]. Intel has proposed a new bus technology, Compute Express Link (CXL) [9]. We believe that the endeavor is continuing.

II. Michrobenchmark design and experiment

pmmeter is a microbenchmarking tool that we have developed to measure the primary performance of basic memory access operations on DRAM and PMEM. The unique feature of pmmeter is in its capability to clarify the performance overhead induced by implicit synchronization options (associated with memory access instructions such as loads and stores) and explicit synchronization instructions (such as barriers and cache invalidation).

pmmeter imposes a synthetic workload on memory space and reports performance metrics of its execution. The user is allowed to configure the workload with a variety of options. The major capability of pmmeter is summarized below. (1) The workload may be executed on memory space organized on DRAM and PMEM separately. pmmerter currently works on Intel Optane DCPMM using the PMDK library; the memory mode and the AppDirect mode are supported. The experiment later presented in this paper only reports the result of the AppDirect mode, since it allows us to observe the bare performance of the device. (2) Data transfer is uni-directional. Load (memory to processors) or store (processors to memory) may be specified. (3) Memory access pattern may be chosen

2375-9356/23/$31.00 ©2023 IEEE
DOI 10.1109/BigComp57234.2023.00069

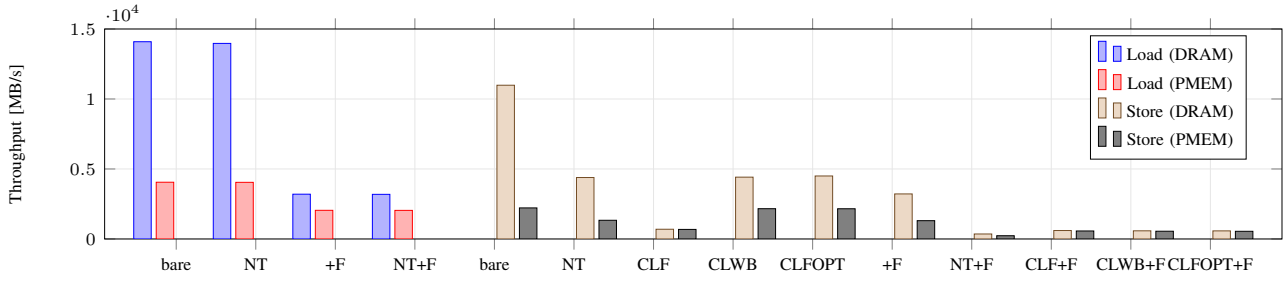

Fig. 1. Average throughput of single-threaded sequential 64-byte loads and stores (2x 32-byte load/store instructions). Throughput of sequential memory access significantly varies with the choice of synchronization options both on DRAM and PMEM. **bare** and **NT** denote a default (temporal) instruction and a non-temporal instruction respectively, whereas **+F** denote a FENCE instruction appended. Additionally, for stores, **CLF**, **CLWB** and **CLFOPT** denote `CLFLUSH`, `CLWB` and `CLFLUSHOPT` instructions appended respectively.

from *sequential* and *random*. The sequential access simply scans the specified memory space from beginning to end, performing loads/stores. In contrast, if the random access is chosen, pmmeter firstly generates an array structure on the space so that each array member stores a reference to another; pmmeter runs the array along with the reference to scatter loads/stores within the space in a random manner. (4) Memory block access size is 64 bytes and instruction unit size is 32 bytes, by default; in this case, each block access is composed of two instructions. These sizes can be manually specified by the user. (5) The temporality of instructions can be specified. By default, the instruction is temporal. But it can be made non-temporal. (6) The cache invalidation is not explicitly enforced by default. One of `CLFLUSH`, `CLWB` and `CLFLUSHOPT` may be appended to each memory access. (7) The barrier is not explicitly enforced by default. `FENCE` may be appended to each memory access. (8) The workload is single-threaded by default. The workload can be made multi-threaded with a specified number of threads. (9) The thread and memory affinity can be optionally specified. For example, the execution thread can be bound to a specific processing core or socket. Similarly, the memory space can be bound to a specific socket.

pmmeter records timestamps by using special instructions such as `RDTSC` while executing the workload, and then calculates the performance metrics from the recorded timestamps. The reported performance metrics include a data transfer throughput, an instruction throughput, and an average latency.

This paper reports a case study of performance measurement that we conducted with pmmeter. A test machine had two Intel Xeon Silver 4215 processors, each having eight cores at 2.5GHz, with 384 GB of DRAM and 1538 GB of Intel Optane DCPMM in total, running CentOS 7.7 with Linux kernel 3.10. The target memory space was allocated to the DRAM and PMEM modules connected to the first socket; PMEM space was configured in the DAX mode. Fig. 1 presents that PMEM performed much slower than DRAM and throughput of sequential memory access significantly varied with the choice of synchronization options on DRAM and PMEM. The figure summarizes the average throughput of single-threaded sequential 64-byte loads and stores (2x 32-byte load/store instructions). PMEM yielded 71.0% to 71.2% lower throughput for loads and 69.6% to 79.8% lower throughput for stores than DRAM. In addition, the explicit call of `FENCE` yielded significant overheads, providing 13.1% to 87.2% rate drops on DRAM and 16.3% to 74.6% rate drops on PMEM.

III. Conclusion

This paper has proposed pmmeter, a microbenchmarking tool for PMEM. pmmeter allows for clarifying the performance impact that the choice of synchronization instructions incurs on a given sequence of memory access. The source code of pmmeter is available at https://github.com/hyoshiok/pmmeter. We would like to extend pmmeter to incorporate a more variety of instruction combinations and to support new PMEM devices other than Intel Optane DCPMM. We hope that pmmeter helps those who want to deeply understand the primary performance property of PMEM.

Acknowledgement

This work has been in part supported by JSPS Grant-in-Aid for Scientific Research (B) 20H04191.

References

[1] Intel, "Intel Optane Persistent Memory," https://www.intel.com/content/www/us/en/products/docs/memorystorage/optane-persistent-memory/optane-dc-persistent-memorybrief.html, 2019.
[2] J. Xu and S. Swanson, "NOVA: A log-structured file system for hybrid volatile/non-volatile main memories," in *Proc. USENIX FAST*, 2016, pp. 323–338.
[3] F. Xia, D. Jiang, J. Xiong, and N. Sun, "Hikv: A hybrid index key-value store for dram-nvm memory systems," in *Proc. USENIX ATC*, 2017, pp. 349–362.
[4] J. Arulraj and A. Pavlo, "How to Build a Non-Volatile Memory Database Management System," in *Proc. ACM SIGMOD*, 2017, p. 1753–1758.
[5] J. Arulraj, M. Perron, and A. Pavlo, "Write-behind logging," *Proc. VLDB Endow.*, vol. 10, no. 4, pp. 337–348, 2016.
[6] Intel, "Intel Optane DC Persistent Memory Readies for Widespread Deployment," https://newsroom.intel.com/news/intel-optane-dc-persistent-memory-readies-widespread-deployment/, 2018.
[7] P. Alcorn, "Intel Kills Optane Memory Business, Pays $559 Million Inventory Write-Off," https://www.tomshardware.com/news/intel-kills-optane-memory-business-for-good, 2022.
[8] Kioxia, "Kioxia Launches Second Generation of High-Performance, Cost-Effective XL-FLASH Storage Class Memory Solution," https://www.kioxia.com/en-jp/business/news/2022/20220802-1.html, 2022.
[9] CXL Consortium, "CXL: Compute Express Link," https://www.computeexpresslink.org/resource-library, 2022.