Optimizing Large-Scale Plasma Simulations on Persistent Memory-based Heterogeneous Memory with Effective Data Placement Across Memory Hierarchy

Jie Ren [1], Jiaolin Luo [1], Ivy Peng [2], Kai Wu [1] and Dong Li [1]
[1] University of California, Merced [2] Lawrence Livermore National Laboratory

1 INTRODUCTION

Plasma simulations are critical for understanding plasma dynamics in space weather and fusion devices. The particle-in-cell (PIC) method is an important model that enables large-scale plasma simulations on high-performance computing (HPC) systems. The PIC method uses computational particles to simulate plasma particles, such as electrons and protons. High-fidelity PIC simulations often use billions and even trillions of particles, which require high memory capacity.

Persistent memory (PM), exemplified by the Intel Optane DC PMM, provides a solution to meet the requirement of high memory capacity in HPC applications. For instance, the Intel Optane PM can provide up to twelve terabyte (TB) memory on a single machine. However, there is a performance gap between PM and DRAM. The Intel Optane NVM has three times latency and about 38% bandwidth of DRAM. Hence, PM often comes with a small DRAM (tens of gigabytes) to boost performance. As a result, PM and DRAM form a heterogeneous memory (HM) system. How to place and migrate data between PM and DRAM to leverage the speed of DRAM and capacity of PM remains active research.

In this study, we use the latest PM hardware to enable large-scale plasma simulations. We analyze the performance and develop performance models for optimizing PIC codes on PM-DRAM systems. Our performance analysis and optimization use a state-of-the-art electromagnetic PIC code called WarpX [1]. Nonetheless, the optimization strategies derived from this work are generally applicable to other PIC-based simulation codes.

WarpX is a mission-critical production-level application designed for efficient executions on large-scale HPC systems and future Exascale machines. As a PIC method, WarpX has high memory footprints for simulating particles moving in electromagnetic fields. A large memory capacity is a key enabler for large-scale simulations in WarpX.

We analysis the performance of WarpX and identify two challenges in optimizing WarpX on PM-based systems. First, WarpX has frequent read/write with a streaming-like access pattern, which intensifies memory accesses. Given the low bandwidth of PM compared to DRAM, this access pattern is unfavorable. Second, the WarpX code uses tens of millions of data objects and frequent memory (de)allocation. Managing such a large number of data objects with diverse properties on DRAM and PM is complex.

We introduce a set of techniques to optimize the performance of WarpX on PM. Data objects are characterized and classified based on their lifetime and memory access patterns. This information guides their placement and migration on PM and DRAM at runtime. We make the best use of memory hierarchy and employ a combination of data migration and processor-cache prefetch mechanisms. The evaluation shows we improved the WarpX execution on Optane-only by 66.4% and outperformed DRAM-cached, the NUMA first-touch policy, and a state-of-the-art HM solution (Nimble [2]) by 38.8%, 45.1% and 83.3%, respectively.

[1]The full paper of this abstract appears in International Conference on Supercomputing (ICS'21) and can be found in http://pasalabs.org/papers/2021/ICS21_WarpX_Optane.pdf

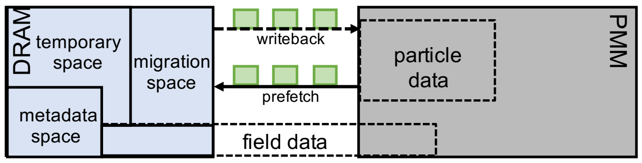

Figure 1: Overview of `WarpX-PM`. The white and shadow boxes represent functionality and mechanisms, respectively.

2 PERFORMANCE CHARACTERIZATION FOR WARPX

PIC codes typically have the following characteristics. PIC codes contain four types of data objects, including `field`, `particle`, `metadata`, and `temporal data`. Field and particles are the main data structures, and consume the most memory footprint. The core PIC routines include four phases – `current deposition`, `field solver`, `field gather`, and `particle pusher`. Each phases access all some of data objects in WarpX.

To study memory access behavior of WarpX, we build profiler for each execution phase in PIC code and collect (1) memory consumption and life time of different data objects, (2) data access pattern for different data objects in different execution phases and (3) memory bandwidth consumption of WarpX's execution. We profile three different types of plasma simulation with WarpX and have the following observations to guide our design:

Observation 1: Particle data and fields data dominate the memory consumption of WarpX. Metadata and temporal data only consumes less than 10% of memory in WarpX.

Observation 2: WarpX is a iterative solver and has streaming-like data access pattern. Among all the execution phases, push particles, deposit current and field solve are the most time-consuming.

Observation 3: The execution of WarpX is not bounded by DRAM/persistent memory bandwidth in most of times.

The memory bandwidth utilization is about 10% of peak memory bandwidth in most of times. Hence, prefetching data to DRAM would not constraint the bandwidth used by the application.

3 `WARPX-PM` DESIGN

Motivated by the above performance characterizations, we introduce `WarpX-PM` (Figure 1), a runtime system to manage data placement on DRAM and PM automatically. WarpX-PM partitions DRAM into four spaces based on functionality and access patterns in

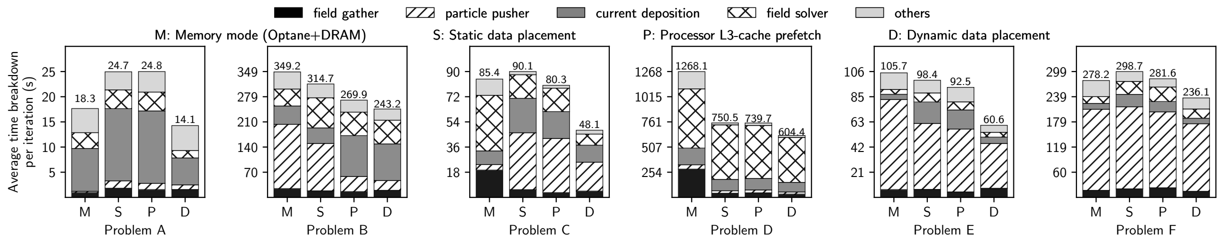

Figure 2: Performance breakdown of main phases of execution to compare the static data placement, cache prefetch, and dynamic placement with memory mode.

WarpX. The metadata space and temporary space reserve memory space in DRAM. Data stored in those space are not causing data movement between DRAM and PMM. The migration space acts as a software-managed DRAM cache to prefetch particles from PM before they are used in computation. Finally, the free space stores the maximum possible field data. We introduce three memory optimization techniques used in different space as follow.

Static Data Placement. The metadata space stores the metadata updated infrequently but accessed frequently. The temporary space stores short-lived data objects frequently allocated and freed. Those short-lived data objects share and reuse the temporary space without causing data movement between DRAM and PMM. WarpX-PM uses static placement to addresses the fundamental limitations in hardware-managed DRAM cache.

Cache Prefetch. The fields are not dynamically migrated between PM and DRAM as particles, because the fields are not contiguously allocated as particles, but spread across the memory address space, which leads to either highly inefficient page-level migration or costly engineering efforts for object-level migration. `WarpX-PM` triggers prefetch before computation by leveraging iterative structures in each execution phase. Specifically, we propose to use a performance model to decide when the prefetch happen to minimize the execution time of accessing the fields.

Dynamic Data Placement. `WarpX-PM` migrates the particles because they are used periodically. To implement the particle prefetching strategy, two challenges must be addressed. First, `WarpX-PM` needs to decide the number of threads to copy the particle batches. `WarpX-PM` uses helper threads instead of application threads to copy particles to avoid delaying the execution of application threads. Using a large number of helper threads accelerate data copy but reduces processor cores and memory bandwidth available for WaprX execution. Using a small number of helper threads increases the risk of exposing data copy into the critical path of WaprX execution if data copy cannot finish in time. Second, the decision of the number of helper threads must be adaptive and lightweight. Different input problems or MPI/OpenMP configurations may consume memory bandwidth differently and need different numbers of helper threads for the best performance. Therefore we build a performance model to generate a particle data migration strategy and minimize the system execution time, by considering the memory bandwidth, the number of helper thread, and computation time of each execution phase. Our performance modeling is lightweight, and we avoid exhaustive search of all possible of data placement by eliminating those that can obviously cause performance loss.

4 EVALUATION

We evaluate `WarpX-PM` on a two socket machine with Intel Xeon Scalable processor. It equips 192GB DRAM and 1.5TB Optane DC PMM. We compare `WarpX-PM` with *memory mode*, which is a hardware-based memory management solution, and *NUMA first-touch*, which is a OS-level memory management solution. The evaluation uses 6 plasma simulations with WarpX which comes from various plasma accelerator simulations with a wide range of memory consumption (**up to 1.2TB**).

Figure 2 reveals that WarpX-PM performs the best in all cases. On average, WarpX-PM outperforms memory mode, Optane-only, and NUMA first-touch by 38.8%, 66.4%, and 45.2%, respectively. We further quantify the performance improvement from static data placement, cache prefetch, and dynamic data placement in `WarpX-PM`. For each problem, we report the per-iteration time in the memory mode as the baseline (M). For comparison, we run `WarpX-PM` with only static data placement (S), with static data placement and cache prefetch (P), and with all the three techniques (D). `WarpX-PM` with the three proposed techniques achieves the best performance in all problems. Different input problems exhibit different sensitivity to these techniques.

We further compare `WarpX-PM` with a state-of-the-art page migration system for HM, named *improved active list* (IAL) [2]. This system improves an existing page replacement mechanism in the Linux kernel (i.e., an FIFO-based active list). `WarpX-PM` outperforms IAL by 83.3% on average and up to 96.6%.

Compared with the memory mode, `WarpX-PM` consumes higher DRAM bandwidth, indicating that fast memory accesses happen more often in `WarpX-PM` to make the best use of DRAM.

We also study the NUMA effects of using PMM on multi-socket platform. Efficient data placement is not only about using DRAM or PM but also about avoiding memory accesses to a remote socket. We compare the memory mode and NUMA first-touch with `WarpX-PM`, and track memory traffic between the two sockets. The evaluation result shows `WarpX-PM` has the lowest intro-socket memory traffic among all cases.

Because of hardware limitation, we cannot evaluate `WarpX-PM` on multiple Optane-based machines. However, `WarpX-PM` focuses on intra-node data movement optimization, and significantly improves performance without impacting communication patterns and application algorithms. We expect that `WarpX-PM` can be applied on larger scales without decreasing application scalability.

REFERENCES

[1] J-L Vay, A Almgren, J Bell, L Ge, DP Grote, M Hogan, O Kononenko, R Lehe, A Myers, C Ng, and others. 2018. Warp-X: A new exascale computing platform for beam–plasma simulations. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 909 (2018), 476–479.
[2] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Nimble Page Management for Tiered Memory Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19).* ACM, New York, NY, USA, 331–345.