Runtime Data Management on Non-Volatile Memory-based High Performance Systems
Kai Wu and Dong Li, EECS, UC Merced

Motivation & Introduction
- Non-volatile memory (NVM) techniques, such as Intel Optane DC PM, provide an enticing combination of performance, capacity, and persistency. However, using NVM to build high performance systems is challenging.
- Problem #1: NVM is significantly slower than DRAM. Using NVM to replace DRAM as main memory to run applications greatly affects the application performance.
  - Pair NVM and DRAM to build a heterogenous memory system (HMS). However, which data of the application should be placed into the limited DRAM of HMS to achieve high performance?
- Problem #2: Ensuring data persistence on NVM is not trivial because other memory components remain volatile and memory writes to NVM are out of order.
  - Programmers need to use assembly instructions (e.g., clflush) to explicitly flush data from the cache to NVM, but frequent cache flushes bring considerable performance overhead to the application.
- In this poster, we propose runtime solutions to efficiently and effectively solve the problem of data placement and data persistence on NVM-based systems.

Data Placement for Task Parallel Programs [2]
We design Tahoe runtime to address the data placement problem for task parallel programs on NVM-based HMS.
- Background Information on Task Parallel Programs
  - Task metadata information
  - Task dependence information
  - Task execution state (Initialized, Ready, Active, Completed)
  - Input/output data object information
  - Task type: Tasks running the same code region with the same input data size have the same task type

Evaluation (use Quartz NVM emulator)
Observation:
- Tahoe reduce execution time by 21% on average (using HMS-oblivious as the baseline).

Data Placement for MPI Programs [1]
We introduce Unimem, a runtime system that directs data placement for MPI-based iterative HPC applications on NVM-based HMS.
- A Preliminary Performance Study for Data Placement for SP from NPB suite
  - A good data placement can effectively bridge the performance gap between DRAM and NVM (31% performance improvement on average).
  - Different data objects manifest different sensitivity to memory bandwidth and latency.
  - Data objects in buffer and out buffer are sensitive to bandwidth, not latency.
  - Data object lhs is sensitive to latency, not bandwidth.

Unimem Design
- Use sampling-based hardware performance counters (performance-based Sampling (PMS) or instruction-based Sampling (IMS))
- Collect the number of last level cache miss event
- Map the event information to data objects via memory addresses
- The performance models estimate performance benefit and data movement cost between NVM and DRAM.
- We trigger data movement only when the benefit outweighs the cost.
- Bandwidth sensitivity vs. latency sensitivity
- Data movement cost can be overestimated by an asynchronous data movement run in parallel.
- Phase local search: Find the local optimal solution based on dynamic programming
  - Cross-phase global search: All phases are treated as a combined single phase to find the optimal data placement.
  - Compare the two searches and choose the better one.

Unimem Runtime
- Phase Profiling
  - Data Placement Decision
  - Optimization Techniques
    - Handling workload variation across iterations
    - Initial data placement
    - Handling large data objects

Evaluation (use Quartz NVM emulator)
Worksloads:
- NAS parallel benchmark suite and Nek5000
- Compare with X-Mem [a pin-based solution, Eurosys '16]
Observation:
- Unimem greatly narrows the performance gap between DRAM only and HMS only.
- The average performance difference between DRAM-only and HMS is only 3% for NVM with 1/2 DRAM bandwidth.
- Unimem performs 10% better than X-mem for Nek5000.

High Performance Cache Line Flushing [3]
We present Ribbon, a runtime system that improves the performance of the cache-line flushing mechanism on NVM to achieve high performance data persistence (no need to modify application semantics).
- Performance Analysis of CLF on NVM (Intel Optane DC PMM)
  - For the seven NVM-aware applications, CLF significantly affects the performance 24%-62%.
  - Concurrent CLF can create resource contention on the hardware buffer inside PM devices and memory controllers, which causes performance loss.

Ribbon Design
- Decoupled Concurrency Control of CLF
  - Ribbon decouples CLF from the application and adjusts the level of CLF concurrency (the number of threads performing CLF adaptively). Ribbon throttles CLF concurrency if contention on PM devices is detected. Conversely, it ramps up CLF concurrency when PM bandwidth is underutilized.
- Proactive Cache Line Flushing
  - Ribbon proactively flushes cache lines to transform cache line in the critical path of the application, which has lower latency than flushing a dirty cache line

Evaluation on Intel Optane DC PMM
Worksloads: PMEMKV, Redis, Level-hashing, Fast&Fair (B+-tree), and Parsec
- Ribbon achieves up to 17.6% (9.3% on average) and up to 49.8% (20.2% on average) improvement of the overall application performance at four app threads (low thread-level parallelism) and 24 app threads (high thread-level parallelism), respectively.

References: