Accelerating the Performance of PMOs Through Prediction and DRAM as Cache (PD-LOaPP)

Derrick Greenspan Naveed Ul Mustafa Andres Delgado Mark Heinrich Yan Solihin

Persistent Memory (PM)

- **Byte Addressable**
- **Non-Volatile/Persistent**
- **High density/cheaper per byte vs. volatile memory (i.e., DRAM)**
- **Performance much closer to volatile memory vs. block storage (i.e., SSD)**
- **Could augment or replace volatile memory as main memory**

Persistent Memory Objects (PMOs)

- Organize PM as a collection of objects (PMOs) holding pointer-rich data structures
- No file backing
- More intuitive than competing designs
- Provides crash consistency, security, and integrity verification

PMO System Calls

Primitive	Description
attach(name,perm,key)	Render accessible the PMO `name`
detach(addr)	Render inaccessible the PMO `addr` points to.
psync(addr)	Force modifications to the PMO to be durable.
pcreate(name,size,key)	Create a PMO `name` of `size` and `key`.
pdestroy(name,key)	Given a valid `key`, delete PMO `name` and reclaim its space.

Challenges with prior work

Poor thread scalability

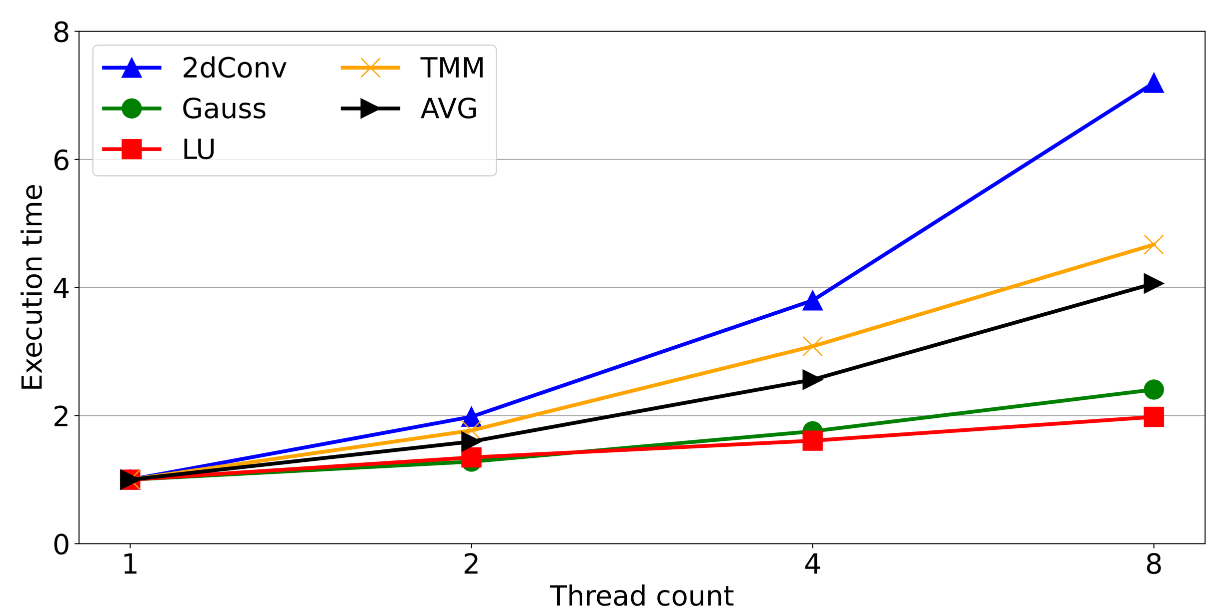

Figure 1. Microbenchmark scalability by thread count

- Additional threads produce diminishing returns
- Product of PM performance characteristics

Decryption within critical path

- Decrypts pages at page fault instead of beforehand
- Write bandwidth of DRAM is much higher than PM

Contributions

1. Proposed PD-LOaPP
2. Presented exploration of PD-LOaPP design space
3. Implemented on Linux kernel with real system running Intel Optane PMem

Design space exploration

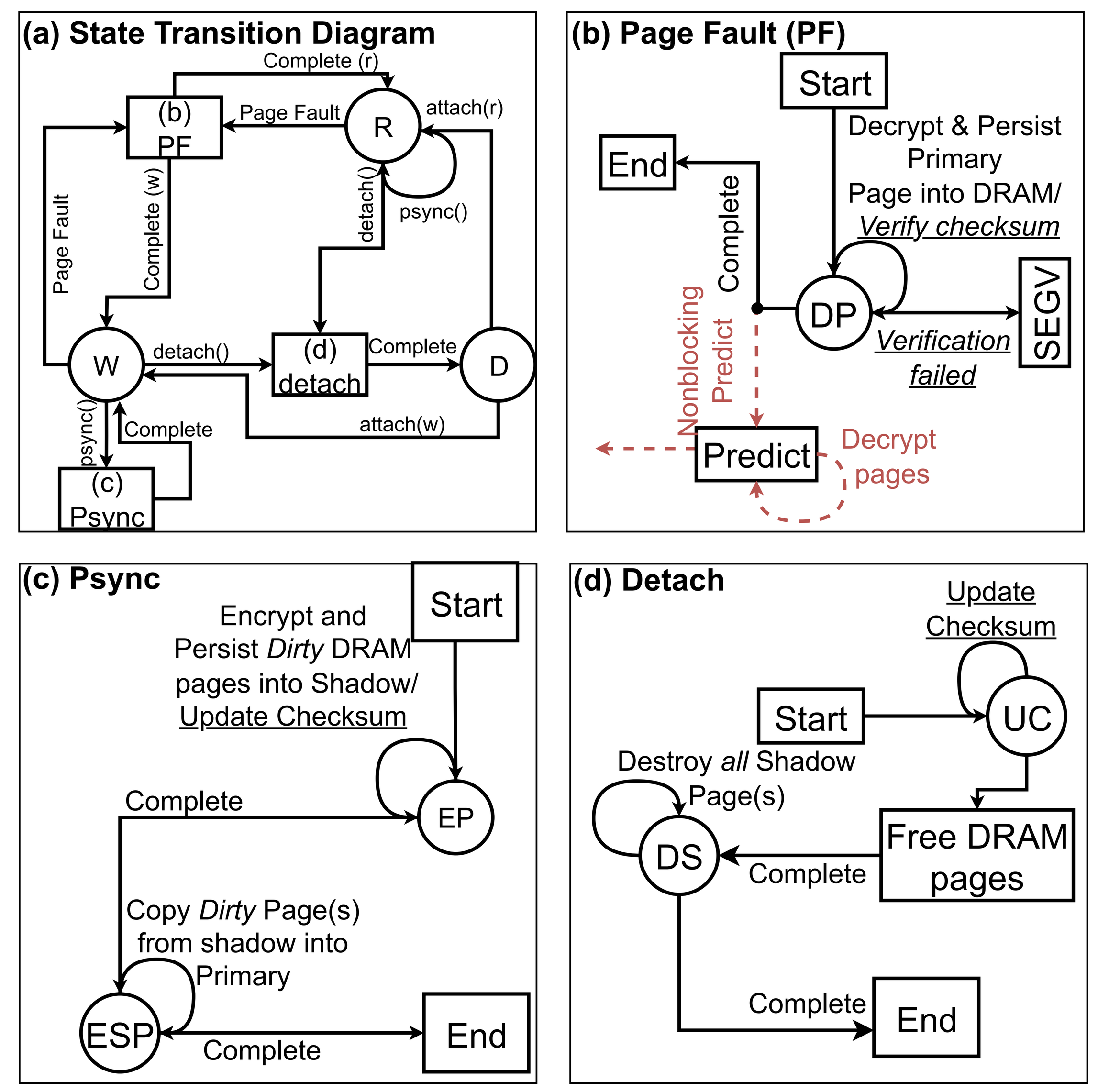

Figure 2. State diagram with DRAM and prediction

DRAM as PM Cache

- Most PM systems have DRAM in addition to PM
- Map pages residing in DRAM into userspace
  - Psync takes slightly longer (copy rather than persist)
    - Psync must be performed infrequently for this not to have a potentially large impact

Per-Page Prediction

- Decrypt pages into DRAM ahead of time
  - Use stream-buffer like design
  - Stream writes directly into DRAM

Implementation

DRAM Page Tracking

- Map page into DRAM at page fault or prediction call
- Serve page from dram via `__get_free_page()`

Prediction

- At page fault time, call prediction handler
- Prediction handler decrypts next X pages, skipping already predicted pages

Evaluation Methodology

- Evaluated two designs: DRAM caching system alone and DRAM with predictor
- Evaluated the impact of prediction depth

Benchmarks

- Microbenchmarks
  - 2dConvolution (256×1024)
  - LU Decomposition (6144)
  - Gaussian Elimination (18432)
  - Tiled Matrix Multiplication $(4096, 16)$
- Filebench
  - Representations of real-world applications
  - File Server (FS), Web Server (WS), Web Proxy (WP), Var Mail (VM)

Evaluation

- IV with prediction is often faster than prior best-case design without IV
- DRAM with prediction is $1.81\times$ faster than the original
- Prediction depth of 8 is best on average

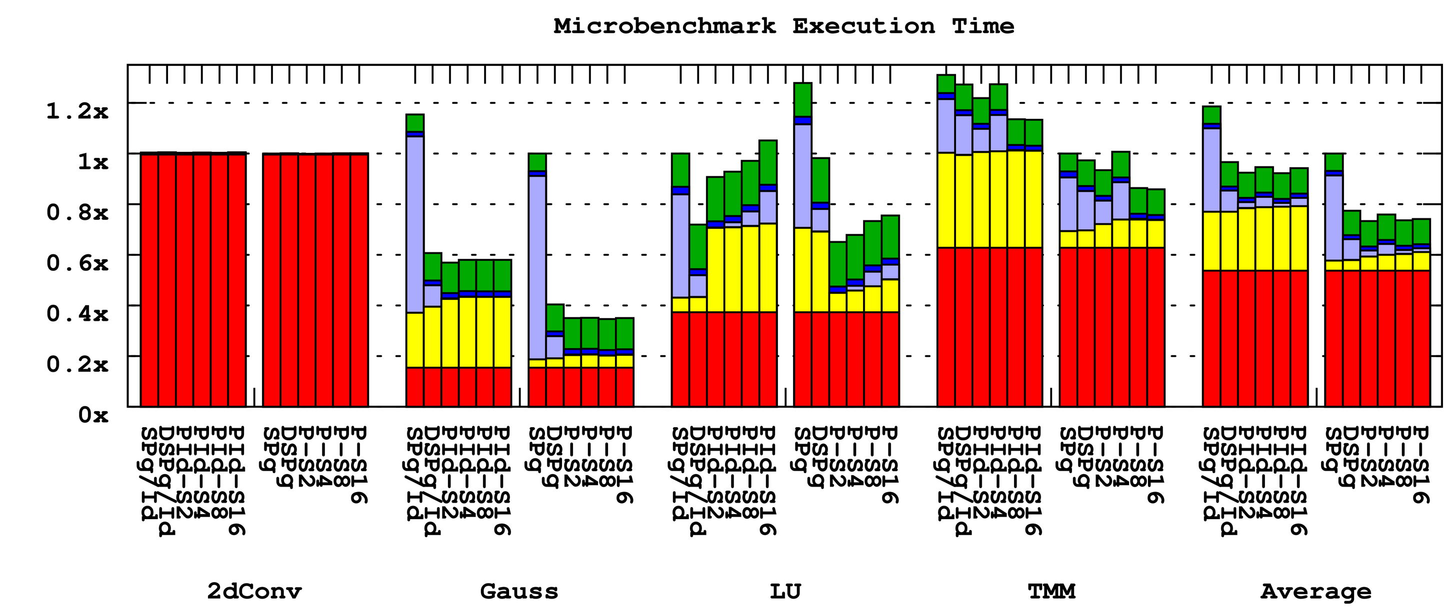

Figure 3. Execution time with and without DRAM prediction

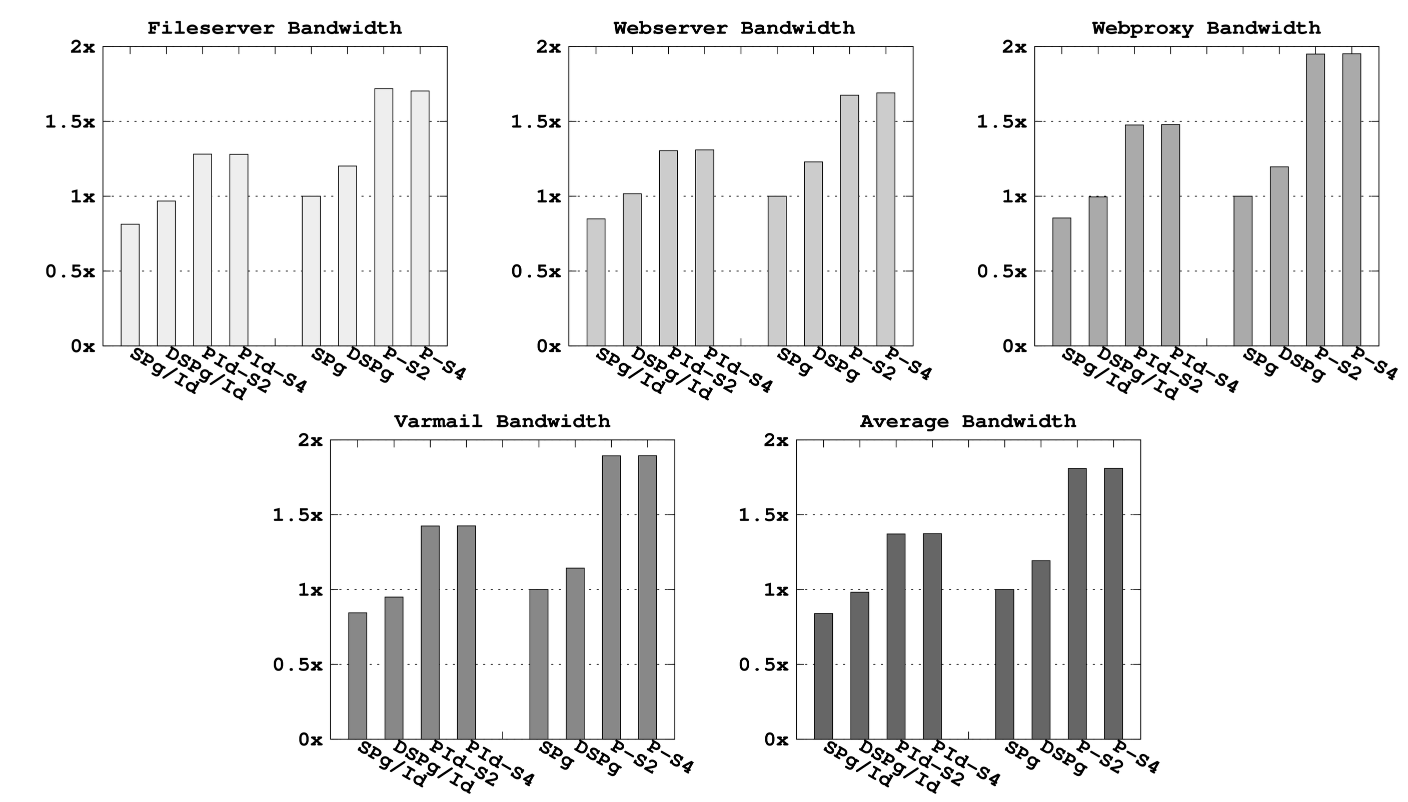

Figure 4. Filebench bandwidth results

Acknowledgements

This work is supported in part by the Office of Naval Research (ONR) under grant N00014-20-1-2750, and the National Science Foundation (NSF) under grant 1900724.