PMEM-Spec: Persistent Memory Speculation
(Strict Persistency Can Trump Relaxed Persistency)

Jungi Jeong and Changhee Jung
Purdue University

Session 6A: Hardware for Crash Consistency
NVMW 2021
Executive Summary

• **Persistency Model** defines persist-orders for failure-recovery

• Challenge for strict persistency ➞ slow!
  • (In general) the more relax, the better performance
  • But relaxing increases programming difficulty
    (like memory consistency)

• **Persistent Memory Speculation**
  • HW/SW codesign for strict persistency
  • 10%~27% speedup compared to relaxed persistency
  • First demonstration of strict persistency outperforming relaxed persistency
Persistent Memory (PM) is Here!

- User-space access to Non-Volatile Memory
- Enables recoverable applications

Intel Optane DC PMM
### PM Programming Challenges

- **PM Stores must be:**
  - Atomic
    - via write-ahead logging*
    - or shadow paging**
    - or idempotent processing$
  
  *(a.k.a. persist-order)*

- **Ordered** (a.k.a. persist-order)
  - flush & fence instructions

---


** ASPLOS 2020

$ MICRO 2018
PM Store Ordering – Strict Persistency

Ex)
- Log A
- Flush & Fence
- Log B
- Flush & Fence
- Data A
- Flush & Fence
- Data B
- Flush & Fence

Flush & Fence for each PM store

Minimal programming burden (compiler-support)
PM Store Ordering – Relaxed Persistency

Ex):
- Log A
- Log B
- **Flush & Fence**
- Data A
- Data B
- **Flush & Fence**

Flush & Fence per epoch

Programming Difficulty (programmer intervention)
PM Store Ordering – Relaxed Persistency

Strict Persistency
- no programming hassle but slow

Relaxed Persistency
- fast but programmer intervention
Related Work*: FENCE Overheads

#1. Reducing FENCE costs

- [MICRO 2016] & [ASPLOS 2017]
- Hiding fence latency
- Delegating the persist-order to HW

#2. Reducing # of FENCEs

* [MICRO2016], [ASPLOS2017], [ISCA2020], [MICRO2020]
Related Work*: FENCE Overheads

1. Reducing FENCE costs
   - [MICRO 2016] & [ASPLOS 2017]
     - Hiding fence latency
     - Delegating the enforcement to HW

2. Reducing # of FENCEs
   - [ISCA 2020]
     - Further relaxing constraints
   - [MICRO 2020]
     - Multiple store paths to PM

* [MICRO2016], [ASPLOS2017], [ISCA2020], [MICRO2020]
Related Work*: HW Complexity

Intra-thread Persist-Order

L1-$\$-side Buffers (PBs)
: governs cache-flush orders based on persistency model

Inter-thread Persist-Order

Cache-coherence extensions
: detects inter-thread dependency

Control Persist-Orders in Private L1-Caches

* [MICRO2016], [ASPLOS2017], [ISCA2020], [MICRO2020]
**Related Work**: Challenge

**L1$\text{-writeback vs. PB Flush**:**
- PB Flush must happen before L1-writeback for a given block

**Solutions of Prior Works**:
- [MICRO2016]: cache-coherent PBs
- [ASPLOS2017]: sticky-bit in LLC for tracking & bloom-filter in PM controllers for checking pending persists in PBs
- [ISCA2020] & [MICRO2020]: buffer L1-writeback

**Prior approaches**:
1. increase HW complexities
2. incur extra latency

* [MICRO2016], [ASPLOS2017], [ISCA2020], [MICRO2020]
**PMEM-Spec**: Persistent Memory Speculation

- Minimal HW changes
- Minimal Program Changes (like Strict Persistency)
- HW/SW Codesign for High Performance Strict Persistency

![Diagram of Persistent Memory Speculation](image_url)
PMEM-Spec Key Ideas

#1. Speculate PM Accesses

- With *Separated* load/store paths to PM

#2. Detect ordering violation (Misspeculation) in HW

- With Arch/Comp interaction

#3. Recover from Misspeculation in SW

- With *failure-atomic* SW as *virtual* power failure
Separated Load/Store Paths to PM

Persist-path: FIFO store path to PM
- Connect SQ to NVM
- Bypass caches
- Drop cache writebacks

Regular-path: Data path through caches
- Serves NVM reads only
What does **PMEM-Spec** Speculate?

**Load Speculation**
- Core → Caches → Ld
- Core → Caches → Ld

: PM load must read latest value from PM

**Store Speculation**
- St → Caches
- Co → St

: PM stores must arrive in the correct order

---

Watch Out for **Ordering Violation** (Misspeculation)!
Load Misspeculation

- **Symptom:** *stale reads*
  - If prior stores are pending in the persist-path
- **Cause:** latency differences in separated load/store paths
Detecting Load Misspeculation

• Observation: if blocks in caches, loads never misspeculate
• Key idea: monitoring *recently evicted* blocks
  • For whether they are overwritten by stores

Q. How long should we monitor?
Detecting Load Misspeculation

- Monitor evicted blocks *until the worst-case persist latency*
  - Fixed in the HW design time*

**Speculation Window**
- Starts on LLC-eviction of dirty blocks (not updating PM data)
- If blocks *being fetched & overwritten* within the window, the fetch was stale

* Best & worst persist-path latencies are determined in HW design time based on HW specification.
Store Misspeculation

- **Cause:** inter-thread dependency
  - Stores to the *same address* from multiple threads

- **Symptom:** *out-of-order persists*

Q. How to capture the store-order between threads? *(without cache-coherence)*

Do not leverage cache-coherence

Thread 0
- ... St A ...
- ...

Thread 1
- St A ...
- ...

Core Cache

PM

Happens-before order

Thread 0
- St A
- ...

Thread 1
- St A
- ...

Core Cache

St A

Thread 0
- St A
- ...

Thread 1
- St A
- ...

Core Cache

St A

Thread 0
- St A
- ...

Thread 1
- St A
- ...

Core Cache

St A
Detecting Store Misspeculation

- Data-Race-Free (DRF) programs
  - Inter-thread dependency *always* happens in critical sections

- Observation:
  - To convey critical section execution order to HW
    - **Speculation ID**
      - global-counter incremented when entering critical section
    - Arriving lower IDs after higher IDs \(\Rightarrow\) out-of-order arrivals
    - New instructions to assign/revoke the speculation ID to a thread (*spec-assign / spec-revoke*)

Critical section execution order == Inter-thread store-order

No Programmer Annotation
  (compiler-generated codes)
Speculation ID Ex) Benign Store

Both stores are benign

Thread 0
Lock
spec-assign
St A
spec-revoke*
Unlock

Thread 1
Lock
spec-assign
St A
spec-revoke*
Unlock

Spec-ID: 101
Spec-ID: 100

PM

* PMEM-Spec untags the speculation ID when exiting critical sections. Please refer to the paper for details.
Speculation ID Ex) **Out-of-Order Persists**

**Thread 0**
Lock
spec-assign
St A
spec-revoke*
Unlock

**Thread 1**
Lock
spec-assign
St A
spec-revoke*
Unlock

* PMEM-Spec untags the speculation ID when exiting critical sections. Please refer to the paper for details.
Misspeculation Recovery

If Misspeculation detected, interrupt the OS
(e.g., synthetic power failure)

Failure-atomic runtime aborts & re-executes TXs
(e.g., Mnemosyne [ASPLOS11], PMDK, iDO [MICRO18])

OS

Runtime

Apps

PMEM-Spec Architecture

Values

Devices

Kernel Space

User Space
Misspeculation Recovery

PMEM-Spec recovers from misspeculation with recovery protocols of failure-atomic SW by treating it as virtual power failure.

If Misspeculation detected, interrupt the OS (e.g., synthetic power failure).

Failure-atomic runtime aborts & re-executes TXs (e.g., Mnemosyne [ASPLOS11], PMDK).
Methodology

- Full system simulation with gem5
  - Linux kernel version: 4.8.13
  - Ubuntu 16.04

<table>
<thead>
<tr>
<th>Processor</th>
<th>8-core, OoO, 2GHz, x86</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 I/D cache</td>
<td>Private, 32/64KB, 4-way, 2ns</td>
</tr>
<tr>
<td>L2 cache</td>
<td>Shared, 16MB, 16-way, 20ns</td>
</tr>
<tr>
<td>PM Controller</td>
<td>32/64-entry read/write queue</td>
</tr>
<tr>
<td>PM</td>
<td>Read: 175ns, write: 94ns</td>
</tr>
<tr>
<td>Persist-Path</td>
<td>20ns</td>
</tr>
</tbody>
</table>

- Comparing schemes
  - Intel X86 (baseline): Epoch Persistency
  - DPO [MICRO’16]: Strict Persistency
  - HOPS [ASPLOS’17]: Epoch Persistency
  - PMEM-Spec: Strict Persistency

- Benchmarks

<table>
<thead>
<tr>
<th>Microbench</th>
<th>Concurrent Queue, Array Swap, HashMap, RB-Tree, TATP, TPCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>WHISPER*</td>
<td>Vacation, Memcached</td>
</tr>
</tbody>
</table>

* S. Nalli et al., ASPLOS 2017.
Evaluation – Throughput

- Microbenchmarks: similar to HOPS (Epoch Persistency)
  - Tiny transactions ➞ less room for speculation

- WHISPER: significantly outperforms previous works
  - Larger transactions ➞ advent speculation opportunities

![Graph showing throughput comparisons between different workloads and systems.](image)

**Throughput**

- **Higher is Better**

<table>
<thead>
<tr>
<th>Workload</th>
<th>IntelX86</th>
<th>DPO</th>
<th>HOPS</th>
<th>PMEM-Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArraySwap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Queue</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HashMap</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RB-Tree</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TATP</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TPCC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vacation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memcached (r50w50)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memcached (r10/r90)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Memcached (r90/w10)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Evaluation – Persist-Path Latency

- Persist-path operations are mostly out of critical paths
- Only at the end of TXs, the persist-path must be drained

Throughput (Normalized to Intel X86)

<table>
<thead>
<tr>
<th>Latency (ns)</th>
<th>HOPS</th>
<th>PMEM-Spec</th>
</tr>
</thead>
<tbody>
<tr>
<td>20</td>
<td>1.2</td>
<td>1.3</td>
</tr>
<tr>
<td>40</td>
<td>1.1</td>
<td>1.3</td>
</tr>
<tr>
<td>60</td>
<td>1.1</td>
<td>1.3</td>
</tr>
<tr>
<td>80</td>
<td>1.1</td>
<td>1.3</td>
</tr>
<tr>
<td>100</td>
<td>1.1</td>
<td>1.3</td>
</tr>
</tbody>
</table>

Higher is Better
Evaluation – Persist-Path Latency

- Persist-path operations are mostly out of critical paths
- Only at the end of TXs, the persist-path must be drained

More in the paper:
- Speculation buffer in the PM controller
- Runtime & OS Support for PMEM-Spec
- Scalability analysis
- More sensitivity analysis
Conclusion: Persistent Memory Speculation

- HW (speculation) / SW (recovery) codesign for **persist-order**

- With separated load/store paths to PM, Misspeculation is *extremely* rare

- Leading to high performance strict persistency outperforming relaxed persistency
PMEM-Spec:
Persistent Memory Speculation
(Strict Persistency Can Trump Relaxed Persistency)

Jungi Jeong and Changhee Jung
Purdue University

Session 6A: Hardware for Crash Consistency
NVMW 2021