Hands-Off Persistence System (HOPS)

Swapnil Haria\textsuperscript{1}, Sanketh Nalli\textsuperscript{1}, Haris Volos\textsuperscript{2}, Kim Keeton\textsuperscript{2}, Mark D. Hill\textsuperscript{1}, Mike M. Swift\textsuperscript{1}

\textsuperscript{1} University of Wisconsin-Madison
\textsuperscript{2} Hewlett Packard Labs
**WHISPER Analysis**

- 4% accesses to PM, 96% to DRAM
- 5-50 epochs/transaction
- Self-dependencies common
- Cross-dependencies rare

**HOPS Design**

- Volatile memory hierarchy (almost) unchanged
- Order epochs without flushing
- Allows multiple copies of same cacheline
- Correct, conservative method based on coherence
Outline

Motivation

HOPS Design

Evaluation
ACID Transactions (currently)

- Acquire Lock
- Prepare Log Entry
- Mutate Data Structure
- Commit Transaction
- Release Lock

FLUSH EPOCH
Base System

CPU 0

Private L1

Shared LLC

Private L1

DRAM Controller

PM Controller

Volatile

Persistent
Base System: Flush

1. Flush A
2. Flush B
Base System: Flush

1. Flush A

2. Flush B

CPU 0

CPU 1

A=1

1. Flush A

DRAM Controller

PM Controller

Volatile

Persistent
Base System: Flush

1. Flush A
2. Flush B

Writeback A

CPU 0
CPU 1

A=1

DRAM Controller
PM Controller

Volatile
Persistent
Base System: Flush

1. Flush A
2. Flush B

Writeback A

DRAM Controller

PM Controller

Volatile
Persistent
Base System: Flush

1. Flush A

CPU 0

DRAM Controller

A=1

Writeback A

CPU 1

PM Controller

2. Flush B

Volatile

Persistent
Base System: Flush

1. Flush A
2. Flush B
Base System: Flush

1. Flush A
2. Flush B

CPU 0

CPU 1

DRAM Controller

PM Controller

A=1

Long Latency PM Write

Volatile

Persistent
Base System: Flush

1. Flush A
2. Flush B

Flush ACK
Base System: Flush

2. Flush B

CPU 0

DRAM Controller

Volatile

A=1

Persistent

CPU 1

PM Controller
Outline

Motivation

HOPS Design

Evaluation
Hands-off Persistence System (HOPS)

- Volatile memory hierarchy (almost) unchanged
- Order epochs without flushing
- Allows multiple copies of same cacheline
- Correct, conservative method for handling cross-dependencies
Base System + Persist Buffers

- CPU
- Private L1
- Persist Buffer Front End
- Shared LLC
- DRAM Controller
- PM Controller
- Persist Buffer Back End
- Volatile
- Persistent

Loads + Stores

Loads

Stores
Persist Buffers

- Volatile buffers
- Front End (per-thread)
  - Address, Ordering Info
- Back End (per-MC)
  - Cacheline data
- Enqueue/Dequeue only
  - Not fully-associative
Hands-off Persistence System (HOPS)

• Volatile memory hierarchy (almost) unchanged

• Order epochs without flushing

• Allows multiple copies of same cacheline

• Correct, conservative method for handling cross-dependencies
OFENCE: Ordering Fence

- Orders stores preceding OFENCE before later stores
Base System + Persist Buffers

Stores

CPU

Shared LLC

Private L1

Persist Buffer Front End

Persist Buffer Front End

Persist Buffer Back End

Loads + Stores

L1 Cache

DRAM Controller

PM Controller

Loads

Stores

Volatile

Persistent
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2

CPU 1

Local TS 25

L1 Cache

A = 1
B = 1

Persist Buffer

B = 1 25
A = 1 25
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFFENCE
5. ST A = 2
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2

CPU 1

Local TS

L1 Cache

Persist Buffer
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFFENCE
5. ST A = 2

CPU 1

Local TS 26

L1 Cache

Persist Buffer

A = 1
B = 1

B = 1 25
A = 1 25
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2
Ordering Epochs without Flushing

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFFENCE
5. ST A = 2
ACID Transactions in HOPS

- Acquire Lock
- Prepare Log Entry \(1^\text{N}\)
- Mutate Data Structure \(1^\text{N}\)
- Commit Transaction
- Release Lock

Volatile Writes
Persistant Writes
OFENCE
ACID Transactions in HOPS

- Acquire Lock
- Prepare Log Entry
- Mutate Data Structure
- Commit Transaction
- Release Lock

Volatile Writes
Persistent Writes
OFENCE
NOT DURABLE!
ACID Transactions in HOPS

- Acquire Lock
- Prepare Log Entry
- Mutate Data Structure
- Commit Transaction
- Release Lock

Volatile Writes
- Persistent Writes

OFENCE
DFENCE
DFENCE: Durability Fence

• Makes the stores preceding DFENCE durable
Durability is important too!

1. $ST\ A = 1$
2. $ST\ B = 1$
3. $LD\ R1 = A$
4. OFENCE
5. $ST\ A = 2$

Diagram:
- CPU 1
- Local TS
- L1 Cache
- Persist Buffer
- A = 2
- B = 1
- A = 2
- 26
Durability is important too!

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2

CPU 1

Local TS

26

A = 2
B = 1

L1 Cache

A = 2
26

Persist Buffer
Durability is important too!

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2

N. DFENCE

CPU 1

Local TS 26

L1 Cache

Persist Buffer

A = 2
B = 1

A = 2 26
Durability is important too!

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2

N. DFENCE
Hands-off Persistence System (HOPS)

• Volatile memory hierarchy (almost) unchanged

• Order epochs without flushing

• Allows multiple copies of same cacheline

• Correct, conservative method for handling cross-dependencies
Preserving multiple copies of cachelines

1. ST A = 1
2. ST B = 1
3. LD R1 = A
4. OFENCE
5. ST A = 2
Hands-off Persistence System (HOPS)

- Volatile memory hierarchy (almost) unchanged
- Orders epochs without flushing
- Allows multiple copies of same cacheline
- Correct, conservative method for handling cross-dependencies
Outline

Motivation

HOPS Design

Evaluation
System Configuration

Evaluated using gem5 full-system mode with the Ruby memory model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU Cores</td>
<td>4 cores, OOO, 2Ghz</td>
</tr>
<tr>
<td>L1 Caches</td>
<td>private, 64 KB, Split I/D</td>
</tr>
<tr>
<td>L2 Caches</td>
<td>private, 2 MB</td>
</tr>
<tr>
<td>DRAM</td>
<td>4GB, 40 cycles read/write latency</td>
</tr>
<tr>
<td>PM</td>
<td>4GB, 160 cycles read/write latency</td>
</tr>
<tr>
<td>Persist Buffers</td>
<td>64 entries</td>
</tr>
</tbody>
</table>
Performance Evaluation

The chart shows the normalized runtime for various operations across different systems and configurations. The x-axis represents different operations: echo, ycsb, redis, ctree, hashmap, vacation, and average. The y-axis represents the normalized runtime, with lower values indicating better performance.

- **x86-64 (NVM)**: Light green bars
- **x86-64 (PWQ)**: Orange bars
- **HOPS (NVM)**: Blue bars
- **HOPS (PWQ)**: Purple bars
- **IDEAL (NON-CC)**: Black bars

Each operation is compared across these configurations to evaluate performance, with the goal of achieving the lowest runtime.
WHISPER Analysis

• 4% accesses to PM, 96% to DRAM
• 5-50 epochs/transaction
• Self-dependencies common
• Cross-dependencies rare

HOPS Design

• Volatile memory hierarchy (almost) unchanged
• Order epochs without flushing
• Allows multiple copies of same cacheline
• Correct, conservative method based on coherence
Questions?

Thanks!