DHTM: Durable Hardware Transactional Memory

Arpit Joshi, Vijay Nagarajan, Marcelo Cintra, Stratis Viglas

ISCA 2018
Persistent Memory is here…
Persistent Memory is here…

Intel Displays 512GB Optane DC Persistent Memory DIMMs

by Paul Alcorn May 31, 2018 at 2:02 AM

Intel held its Memory and Storage day today at its Santa Clara headquarters to announce its Optane DC Persistent Memory DIMMs. The new DIMMs slot into the DRAM interface, just like a normal stick of RAM, but come in three capacities of 128, 256, and 512GB. That’s a massive capacity increase compared to the industry-leading 128GB DDR4 memory sticks. Intel designed the DIMMs to bridge both the performance and pricing gap between storage and memory, so the new DIMMs should land at much lower price points than typical DRAM.

Intel Launches Optane DIMMs Up To 512GB: Apache Pass Is Here!

by Ian Cutress & Billy Tallis on May 30, 2018 2:15 PM BST

Intel teases Optane DIMMS, but you may need a new Xeon first

128GB, 256GB and 512GB modules offered as new storage tier below RAM, above SSD

By Simon Sharwood, APAC Editor 31 May 2018 at 03:50

Intel’s new Optane DC persistent memory DIMM. (Credit: AnandTech)
Persistent Memory Systems

L1

LLC

Persistent Memory
Persistent Memory Systems

- **Persistent Memory**
  - Non-volatility over the memory bus
  - Load/Store interface to persistent data
Persistent Memory Systems

- **Persistent Memory**
  - Non-volatility over the memory bus
  - Load/Store interface to persistent data

System Crashes
Persistent Memory Systems

- **Persistent Memory**
  - Non-volatility over the memory bus
  - Load/Store interface to persistent data

- **Crash Consistency**
  - Is the persistent state consistent?
  - Programming Model: ACID Transactions
Persistent Memory Systems

- Persistent Memory - Non-volatility over the memory bus - Load/Store interface to persistent data

- Crash Consistency
  - Is the persistent state consistent?
  - Programming Model: ACID Transactions

“Ensuring failure atomicity for all this computation without failure-atomic transactions is practically infeasible, if not impossible.”

Marathe et al. [HotStorage’17]
Persistent Memory Systems

• Persistent Memory
  - Non-volatility over the memory bus
  - Load/Store interface to persistent data

• Crash Consistency
  - Is the persistent state consistent?
  - Programming Model: ACID Transactions

“Ensuring failure atomicity for all this computation without failure-atomic transactions is practically infeasible, if not impossible.”

Marathe et al. [HotStorage’17]

How fast can we support ACID?
ACID Transactions

L1

LLC

L1

Persistent Memory
ACID Transactions

Atomic Visibility

Persistent Memory
ACID Transactions

Atomic Visibility

Persistent Memory

Atomic Durability
ACID Transactions

Persistent Memory

Atomic Visibility

- Locks
- STM
- HTM

Atomic Durability
ACID Transactions

![Diagram showing L1, LLC, Persistent Memory, Atomic Visibility, Locks, STM, HTM, Checkpointing, S/W Logging, H/W Logging, and Atomic Durability]
ACID Transactions

L1 \rightarrow \text{LLC} \rightarrow \text{Persistent Memory}

Atomic Visibility

Locks \rightarrow STM \rightarrow HTM

Check-pointing \rightarrow S/W Logging \rightarrow H/W Logging

Atomic Durability
Atomic Visibility: HTM
Atomic Visibility: HTM

- Commercial HTMs [Intel, IBM]

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>B = 20</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Atomic Visibility: HTM

- **Commercial HTMs** [Intel, IBM]
  - **Version Management**: read/write sets in L1 cache

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>B = 20</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Atomic Visibility: HTM

- **Commercial HTMs** [Intel, IBM]
  - **Version Management**: read/write sets in L1 cache
  - **Conflict Detection**: piggy back on the coherence protocol

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>B = 20</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>
Atomic Visibility: HTM

- **Commercial HTMs** [Intel, IBM]
  - **Version Management**: read/write sets in L1 cache
  - **Conflict Detection**: piggy back on the coherence protocol
  - **Commit**: make updates non-speculative

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td>Y</td>
</tr>
<tr>
<td>B = 20</td>
<td>Y</td>
<td></td>
</tr>
</tbody>
</table>
Atomic Visibility: HTM

- **Commercial HTMs** [Intel, IBM]
  - **Version Management**: read/write sets in L1 cache
  - **Conflict Detection**: piggy back on the coherence protocol
  - **Commit**: make updates non-speculative
  - **Abort**: invalidate write set
Atomic Visibility: HTM

- **Commercial HTMs** [Intel, IBM]
  - **Version Management**: read/write sets in L1 cache
  - **Conflict Detection**: piggy back on the coherence protocol
  - **Commit**: make updates non-speculative
  - **Abort**: invalidate write set

Write-sets in commercial HTMs limited by the size of the L1 cache.
Atomic Durability: Logging
Atomic Durability: Logging

- **Logging for durability** [Doshi’16, Joshi’17, Shin’17, Ogleari’18]

<table>
<thead>
<tr>
<th>Persistent Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>In-place Values</td>
</tr>
<tr>
<td>A = 10</td>
</tr>
<tr>
<td>B = 20</td>
</tr>
<tr>
<td>C = 30</td>
</tr>
</tbody>
</table>
Atomic Durability: Logging

- Logging for durability [Doshi’16, Joshi’17, Shin’17, Ogleari’18]
  - Write a log entry for every update
Atomic Durability: Logging

- **Logging for durability** [Doshi’16, Joshi’17, Shin’17, Ogleari’18]
  - Write a log entry for every update
  - **Commit**: Update the values in-place
Atomic Durability: Logging

• **Logging for durability** [Doshi’16, Joshi’17, Shin’17, Ogleari’18]
  - Write a log entry for every update
  - **Commit**: Update the values in-place
  - **Abort**: Undo any in-place updates

<table>
<thead>
<tr>
<th>Persistent Memory</th>
<th>In-place Values</th>
<th>Transaction Log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A = 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B = 20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C = 30</td>
<td></td>
</tr>
</tbody>
</table>
Atomic Durability: Logging

• **Logging for durability** [Doshi’16, Joshi’17, Shin’17, Ogleari’18]
  - Write a log entry for every update
  - **Commit**: Update the values in-place
  - **Abort**: Undo any in-place updates

<table>
<thead>
<tr>
<th>Persistent Memory</th>
<th>In-place Values</th>
<th>Transaction Log</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A = 10</td>
<td></td>
</tr>
<tr>
<td></td>
<td>B = 20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>C = 30</td>
<td></td>
</tr>
</tbody>
</table>

× In-place updates in the critical path of commit
× High memory write bandwidth requirement
ACID = HTM + Logging

Goals:

- Support fast commits
- Minimise memory bandwidth consumption
- Extend the supported transaction size
- Maintain the simplicity of commercial HTMs
DHTM: Durable Hardware Transactional Memory

- L1
- LLC
- Log Writes
- Persistent Memory
DHTM: Durable Hardware Transactional Memory

Commercial HTM + Hardware Redo Log
DHTM: Durable Hardware Transactional Memory

Commercial HTM + Hardware Redo Log
- H/W Redo Log + Log Buffer
  - Reduced memory bandwidth
  - Fast commits
DHTM: Durable Hardware Transactional Memory

- H/W Redo Log + Log Buffer
  - Reduced memory bandwidth
  - Fast commits
- H/W Log + Sticky State
  - Extended transaction size to the LLC
  - Simplicity of commercial HTM
DHTM: Log Buffer
DHTM: Log Buffer

- Redo Log Bandwidth Problem
DHTM: Log Buffer

- Redo Log Bandwidth Problem
  - write a log entry for every store
DHTM: Log Buffer

- **Redo Log Bandwidth Problem**
  - write a log entry for every store
  - multiple stores create multiple log entries
DHTM: Log Buffer

- **Redo Log Bandwidth Problem**
  - write a log entry for every store
  - multiple stores create multiple log entries

- **Solution: Log Buffer**
DHTM: Log Buffer

- **Redo Log Bandwidth Problem**
  - write a log entry for every store
  - multiple stores create multiple log entries

- **Solution: Log Buffer**
  - track cache lines being modified
DHTM: Log Buffer

- **Redo Log Bandwidth Problem**
  - write a log entry for every store
  - multiple stores create multiple log entries

- **Solution: Log Buffer**
  - track cache lines being modified
  - multiple writes coalesced in a log entry
DHTM: Log Buffer

- **Redo Log Bandwidth Problem**
  - write a log entry for every store
  - multiple stores create multiple log entries

- **Solution: Log Buffer**
  - track cache lines being modified
  - multiple writes coalesced in a log entry
  - log entry written to persistent memory on eviction from log buffer
DHTM: Transaction States
DHTM: Transaction States

Begin Transaction

Active
DHTM: Transaction States

- Begin Transaction
- Active
- End Transaction & Log Records Persisted
- Commit
DHTM: Transaction States

Begin Transaction → Active

End Transaction & Log Records Persisted → Commit

In-place Data Persisted → Commit Complete
DHTM: Transaction States

- **Active**: Begin Transaction, End Transaction & Log Records Persisted
- **Commit**: In-place Data Persisted
- **Abort**: Conflict
- **Commit Complete**:
DHTM: Commit Example

**L1 Cache**

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State**

**Log Buffer**

**Persistent Memory**

**In-place Values**

<table>
<thead>
<tr>
<th>A = 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>B = 20</td>
</tr>
<tr>
<td>C = 30</td>
</tr>
</tbody>
</table>

**Transaction Log**

**Begin_Transaction**

- Write (A=15)
- Read (B)
- Write (B=25)

**End_Transaction**
DHTM: Commit Example

**L1 Cache**

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**State**

**Active**

**Log Buffer**

**Persistent Memory**

**In-place Values**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td>10</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td>20</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>30</td>
</tr>
</tbody>
</table>

**Transaction Log**

**Begin_Transaction**

- Write (A=15)
- Read  (B)
- Write (B=25)

**End_Transaction**
DHTM: Commit Example

**L1 Cache**

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A = 15</strong></td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**State**

- Active

**Log Buffer**

- **A**

**Persistent Memory**

<table>
<thead>
<tr>
<th>In-place Values</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A = 10</strong></td>
</tr>
<tr>
<td><strong>B = 20</strong></td>
</tr>
<tr>
<td><strong>C = 30</strong></td>
</tr>
</tbody>
</table>

**Transaction Log**

**Begin_Transaction**

- Write (A=15)
- Read (B)
- Write (B=25)

**End_Transaction**
DHTM: Commit Example

L1 Cache

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>B = 20</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

State

Active

Log Buffer

A

Persistent Memory

In-place Values

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 10</td>
<td></td>
</tr>
<tr>
<td>B = 20</td>
<td></td>
</tr>
<tr>
<td>C = 30</td>
<td></td>
</tr>
</tbody>
</table>

Transaction Log

Begin_Transaction

Write (A=15)

Read (B)

Write (B=25)

End_Transaction
DHTM: Commit Example

**L1 Cache**

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>B = 25</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

**State**

**Active**

**Log Buffer**

**Persistent Memory**

**In-place Values**

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 10</td>
</tr>
<tr>
<td>B = 20</td>
</tr>
<tr>
<td>C = 30</td>
</tr>
</tbody>
</table>

**Transaction Log**

<table>
<thead>
<tr>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
</tr>
</tbody>
</table>

**Begin_Transaction**

- Write (A=15)
- Read (B)
- Write (B=25)

**End_Transaction**
**DHTM: Commit Example**

---

**L1 Cache**

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>B = 25</td>
<td></td>
<td>1</td>
</tr>
</tbody>
</table>

**State**

- Commit

**Log Buffer**

---

**Persistent Memory**

**In-place Values**

<table>
<thead>
<tr>
<th>Value</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 10</td>
<td></td>
</tr>
<tr>
<td>B = 20</td>
<td></td>
</tr>
<tr>
<td>C = 30</td>
<td></td>
</tr>
</tbody>
</table>

**Transaction Log**

- A = 15
- B = 25
- Commit

---

**Begin_Transaction**

Write (A=15)
Read (B)
Write (B=25)

**End_Transaction**
DHTM: Commit Example

L1 Cache

<table>
<thead>
<tr>
<th>Cache Line</th>
<th>R</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B = 25</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

State

Log Buffer

Commit Complete

Persistent Memory

In-place Values

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
<td></td>
<td></td>
</tr>
<tr>
<td>B = 25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>C = 30</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Transaction Log

<p>| |</p>
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A = 15</td>
</tr>
<tr>
<td>B = 25</td>
</tr>
<tr>
<td>Commit</td>
</tr>
<tr>
<td>Complete</td>
</tr>
</tbody>
</table>

Begin_Transaction

Write (A=15)
Read (B)
Write (B=25)
End_Transaction
DHTM: Supporting Overflow
DHTM: Supporting Overflow

• Problems with Overflow:
DHTM: Supporting Overflow

• Problems with Overflow:
  - Version Management:
    - global operation on write-set on a commit/abort
    - overhead infeasible in larger caches (beyond L1)
DHTM: Supporting Overflow

• Problems with Overflow:
  - Version Management:
    - global operation on write-set on a commit/abort
    - overhead infeasible in larger caches (beyond L1)
  - Conflict Detection:
    - additional metadata to detect conflicts
    - increased complexity due to NACK based protocols
DHTM: Supporting Overflow
DHTM: Supporting Overflow

• Solution
DHTM: Supporting Overflow

• Solution
  - Version Management:
  - Overflow List
DHTM: Supporting Overflow

**Solution**
- Version Management:
  - Overflow List
DHTM: Supporting Overflow

• Solution
  - Version Management:
  - Overflow List
DHTM: Supporting Overflow

• Solution
  - Version Management:
    - Overflow List
  - Conflict Detection:
    - maintain sticky state on overflow (similar to LogTM)
    - avoid NACK by restricting overflow to LLC
DHTM: Supporting Overflow

• Solution
  - Version Management:
    - Overflow List
  - Conflict Detection:
    - maintain sticky state on overflow (similar to LogTM)
    - avoid NACK by restricting overflow to LLC

Further details on supporting overflows are in the paper.
Evaluation

<table>
<thead>
<tr>
<th>Atomic Visibility</th>
<th>Atomic Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOM</td>
<td>Locks</td>
</tr>
<tr>
<td>LogTM+ATOM</td>
<td>HTM (LogTM)</td>
</tr>
<tr>
<td>DHTM</td>
<td>HTM</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Atomic Visibility</th>
<th>Atomic Durability</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATOM</td>
<td>Hardware Undo Log</td>
</tr>
<tr>
<td>LogTM+ATOM</td>
<td>Hardware Undo Log</td>
</tr>
<tr>
<td>DHTM</td>
<td>Hardware Redo Log (Log Buffer)</td>
</tr>
</tbody>
</table>

• **System Configuration**
  - We evaluate an 8-core machine with a 2-level cache hierarchy
  - HTM’s implement (first) writer wins conflict resolution policy
Evaluation
Evaluation

![Graph showing evaluation results for various data structures.](image-url)
Evaluation

![Bar Chart]

- **ATOM**
- **LogTM+ATOM**
- **DHTM**

- queue
- hash
- sdg
- sps
- btree
- rbtree
- gmean
Evaluation

- ATOM
- LogTM+ATOM
- DHTM

Comparison of performance metrics for different data structures:
- Queue
- Hash
- SDG
- SPS
- Btree
- RBtree
- Gmean

26% improvement observed in some cases.
Evaluation

![Bar chart showing performance comparison between ATOM, LogTM+ATOM, and DHTM for different data structures. The gmean with DHTM shows a 17% improvement over ATOM and LogTM+ATOM.](image-url)
Conclusion

- Persistent memory systems require crash consistency
- ACID Transactions: widely understood crash consistency mechanism
- DHTM: ACID transactions in hardware
  - Atomic Visibility: commercial HTM
  - Atomic Durability: bandwidth optimized hardware redo log
  - Leverage hardware logging to extend transaction size unto LLC
DHTM: Durable Hardware Transactional Memory

Arpit Joshi, Vijay Nagarajan, Marcelo Cintra, Stratis Viglas

ISCA 2018