Efficient Persist Barriers for Multicores

Arpit Joshi, Vijay Nagarajan, Marcelo Cintra, Stratis Viglas
Summary

• Efficient **persist barrier**

• Used to **implement persistency models**

  • **Persistency** = when stores become **durable**
    (Consistency = when stores become visible)

• Evaluated

  • **Buffered Epoch Persistency**

  • **Bulk Strict Persistency**
Persistent Memory

Access Granularity

Access Latency
Persistent Memory

Access Granularity

Access Latency

Cache

DRAM

Secondary Storage
Persistent Memory

Access Latency

Access Granularity

Cache
DRAM
NVRAM

PCM
STT-MRAM
3D Xpoint

Secondary Storage
Persistent Memory

Fast, fine grained persistence.

Access Latency

Persistent Memory Components:
- PCM
- STT-MRAM
- 3D Xpoint

Memory Hierarchy:
- Cache
- DRAM
- NVRAM
- Secondary Storage
Persistent Memory

**Advantages:**
- Unify memory and storage
- Access to persistent data through processor load/store interface

**Challenge:**
- Maintaining consistency of data structures in memory
Consistency Challenge

Core

Cache

DRAM

Software Controlled

Secondary Storage
Consistency Challenge

Core

Cache

DRAM

Secondary Storage

Core

Cache

NVRAM

Secondary Storage

Hardware Controlled

Software Controlled
Linked List - Naïve

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Update Head Pointer
Linked List - Naïve

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Update Head Pointer

Cache

Node 0 → Node 1 → Node 2

HEAD

NVRAM

December 9, 2015
MICRO-48
Linked List - Naïve

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Update Head Pointer
Linked List - Naïve

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Update Head Pointer
Linked List - Naïve

System Crash!
**Linked List - Failsafe**

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Persist Barrier
4. Update Head Pointer
Linked List - Failsafe

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Persist Barrier
4. Update Head Pointer
Linked List - Failsafe

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Persist Barrier
4. Update Head Pointer
Linked List - Failsafe

Pseudo-code

1. Create Node
2. Update Node Pointer
3. **Persist Barrier**
4. Update Head Pointer

Diagram:

- **Cache**
  - HEAD
  - Node 0
  - Node 1
  - Node 2

- **NVRAM**
  - HEAD
  - Node 0
  - Node 1
  - Node 2
Linked List - Failsafe

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Persist Barrier
4. Update Head Pointer
Linked List - Failsafe

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Persist Barrier
4. Update Head Pointer

Epoch A
Epoch B
Linked List - Failsafe

Pseudo-code

1. Create Node
2. Update Node Pointer
3. Persist Barrier
4. Update Head Pointer

Programmer Introduced
Linked List - Failsafe

**Pseudo-code**

1. Create Node
2. Update Node Pointer
3. Persist Barrier
4. Update Head Pointer

Divide program execution into epochs through programmer inserted persist barriers = Epoch Persistence

Programmer Introduced
Epoch Persistence*

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Epoch Persistence*

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Epoch Persistence*

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Epoch Persistence*

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Epoch Persistence*

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Epoch Persistence*

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Epoch Persistence*

Persist operations happen in the critical path of execution.

* Pelley et. al., "Memory Persistency", in ISCA-2014.
Buffered Epoch Persistence* through Lazy Barrier (LB)

• Implementation of Epoch Persistence

• Durability lags visibility

  • To allow performing persist operations out of critical path

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Buffered Epoch Persistence* through Lazy Barrier (LB)

Visibility

------------

Persistence

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Buffered Epoch Persistence* through Lazy Barrier (LB)

Epoch 1

Visibility

Persistence

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Buffered Epoch Persistence* through Lazy Barrier (LB)

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Buffered Epoch Persistence* through Lazy Barrier (LB)

Visibility

```
Epoch 1
a b c a
e
```

```
Epoch 2
d e d
```

Persistence

```
| b |
```

Cache Line Eviction

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Buffered Epoch Persistence* through Lazy Barrier (LB)

* Pelley et. al., “Memory Persistency”, in ISCA-2014.
Buffered Epoch Persistence* through Lazy Barrier (LB)

* Pelley et. al., “Memory Persistency”, in ISCA-2014.

Conflicts bring persist operations back in the critical path.
Intra-thread Conflict
Intra-thread Conflict

Visibility

Persistence
Proactive Flush (PF)

- **Persist Triggers in Lazy Barrier**
  - **Passive Trigger:** cache line eviction
  - **Reactive Trigger:** flush on (intra/inter)-thread conflicts
Proactive Flush (PF)

- Persist Triggers in Lazy Barrier
  - Passive Trigger: cache line eviction
  - Reactive Trigger: flush on (intra/inter)-thread conflicts

- Persist Trigger with Proactive Flush
  - Proactive Trigger: proactively flush on epoch completion
Proactive Flush (PF)

Visibility

--------------

Persistence
Proactive Flush (PF)

Visibility

| a | b | c | a |

Persistence

b

Epoch 1
Proactive Flush (PF)

Visibility

```
| a | b | c | a | d | e | d |
```

Persistence

```
| b |
| a |
| c |
```

Epoch 1

Epoch 2
Proactive Flush (PF)

Visibility

Epoch 1

Epoch 2

Epoch 3

Persistence
Proactive Flush (PF)
Proactive Flush (PF)

Visibility

Epoch 1

a b c a d e d p q d

Epoch 2

Epoch 3

Persistence

Proactive Flush

Proactive Flush
Proactive Flush (PF)

Visibility

a b c a d e d p q d

Persistence

b a c d e

Proactive Flush

Reduces the probability of encountering conflicts.
Inter-thread Conflict

Thread $T_0$
Visibility

Persistence

Thread $T_1$
Visibility
Inter-thread Conflict

Thread $T_0$
Visibility

$\begin{array}{cccc}
\text{W}_A & \text{W}_B & \text{W}_E & \text{W}_F \\
\end{array}$


Persistence

Thread $T_1$
Visibility

$\begin{array}{ccc}
\text{R}_X & \text{R}_Y & \text{W}_Z \\
\end{array}$
Inter-thread Conflict

Thread $T_0$
Visibility

$W_A$ $W_B$ $W_E$ $W_F$

Persistence

Thread $T_1$
Visibility

$R_X$ $R_Y$ $W_Z$ $R_P$ $R_B$

Epoch $E_{00}$

Epoch $E_{10}$
Inter-thread Conflict

Thread $T_0$
Visibility

$W_A$ $W_B$ $W_E$ $W_F$

Persistence

$A$ $B$ $E$ $F$

Thread $T_1$
Visibility

$R_X$ $R_Y$ $W_Z$ $R_P$ $R_B$

Epoch $E_{00}$

Epoch $E_{10}$
Epoch $E_{11}$
Inter-thread Conflict

Thread $T_0$
- Visibility
  - $W_A, W_B, W_E, W_F$
  - Epoch $E_{00}$

Persistence
- $A, B, E, F$
- $Z$

Thread $T_1$
- Visibility
  - Epoch $E_{10}$
  - Epoch $E_{11}$

December 9, 2015
MICRO-48
Inter-thread Dependency Tracking (IDT)

- Lazy barrier
  - No tracking of inter-thread dependencies
  - Need to enforce dependencies online
Inter-thread Dependency Tracking (IDT)

• Lazy barrier
  • No tracking of inter-thread dependencies
  • Need to enforce dependencies online

• Inter-thread Dependency Tracking
  • Add inter-thread dependence tracking registers
  • Track dependencies to enforce offline
Inter-thread Dependency Tracking (IDT)

<table>
<thead>
<tr>
<th>Thread $T_0$</th>
<th>Visibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistence</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Thread $T_1$</th>
<th>Visibility</th>
</tr>
</thead>
</table>

**IDT Table**

<table>
<thead>
<tr>
<th>Source</th>
<th>Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Inter-thread Dependency Tracking (IDT)

<table>
<thead>
<tr>
<th>IDT Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
</tr>
</tbody>
</table>

Thread T₀
Visibility

Epoch E₀₀

Wₐ W₇ W₈ W₉

Persistenct

Thread T₁
Visibility

Epoch E₁₀ Epoch E₁₁

ₐ W₇ W₈ W₉

Rₙ Rₕ W₃ Rₚ Rₗ
Inter-thread Dependency Tracking (IDT)

<table>
<thead>
<tr>
<th>IDT Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source</td>
</tr>
<tr>
<td>Epoch E00</td>
</tr>
</tbody>
</table>

Thread T₀
Visibility

Thread T₁
Visibility

Epoch E₀₀

Epoch E₁₀
Epoch E₁₁
Inter-thread Dependency Tracking (IDT)

Thread $T_0$
- Visibility
  - $W_A$, $W_B$, $W_E$, $W_F$

Persistence
- $A$, $B$, $E$, $F$

Thread $T_1$
- Visibility

Epochs
- $E_{00}$
- $E_{10}$
- $E_{11}$

IDT Table

<table>
<thead>
<tr>
<th>Source</th>
<th>Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epoch E00</td>
<td>Epoch E11</td>
</tr>
</tbody>
</table>

December 9, 2015
Inter-thread Dependency Tracking (IDT)

Thread $T_0$

Visibility:

- $W_A$
- $W_B$
- $W_E$
- $W_F$

Persistence:

- $A$
- $B$
- $E$
- $F$

- $Z$

Thread $T_1$

Visibility:

- $R_X$
- $R_Y$
- $W_Z$
- $R_P$
- $R_B$
- $R_Q$
- $W_E$

Epochs:

- $E_{00}$
- $E_{10}$
- $E_{11}$

IDT Table:

<table>
<thead>
<tr>
<th>Source</th>
<th>Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epoch E00</td>
<td>Epoch E11</td>
</tr>
</tbody>
</table>
Inter-thread Dependency Tracking (IDT)

Reduces the latency of conflicting requests.

IDT Table

<table>
<thead>
<tr>
<th>Source</th>
<th>Dependent</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epoch E00</td>
<td>Epoch E11</td>
</tr>
</tbody>
</table>
Evaluation

<table>
<thead>
<tr>
<th>Design</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LB</td>
<td>Lazy barrier</td>
</tr>
<tr>
<td>LB+IDT</td>
<td>Lazy barrier with inter-thread dependence tracking (IDT)</td>
</tr>
<tr>
<td>LB+PF</td>
<td>Lazy barrier with proactive flush (PF)</td>
</tr>
<tr>
<td>LB++</td>
<td>Lazy barrier with both IDT and PF</td>
</tr>
</tbody>
</table>

Persist Barrier Designs

- **Persistency Models**
  - **Buffered Epoch Persistency** (BEP)
    - maintaining in-memory persistent data structures
  - **Bulk Strict Persistency** (BSP) = BEP + atomicity
    - provide stronger persistency model (strict persistency) — similar to doing sequential consistency in bulk mode*

System Configuration

• We evaluate proposed design using GEM5 full-system simulation mode

• 32 Core CMP with 32x1MB LLC cache banks and 4 memory controllers
  • More details on implementation of persist barrier for such a system are in the paper.
BEP
Transaction Throughput

Higher is Better
BEP
Transaction Throughput

Higher is Better

3%
BEP
Transaction Throughput

Higher is Better
BEP
Transaction Throughput

Higher is Better

22%
BSP

Execution Time

Lower is Better

<table>
<thead>
<tr>
<th>Normalized execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td>canneal</td>
</tr>
<tr>
<td>LB</td>
</tr>
</tbody>
</table>

Lower is Better
BSP

Execution Time

Lower is Better
BSP
Execution Time

Lower is Better
Conclusion

• We propose an efficient implementation of a persist barrier primitive
  • Buffered implementation, to move persists out of critical path
  • We highlight how conflicts bring them back into critical path
• We propose and implement two optimizations
  • Proactive Flush: Reduce the percentage of conflicting epochs
  • Inter-thread Dependence Tracking: Reduce the penalty of inter-thread conflicts
• We demonstrate the efficacy by implementing two persistence models, namely BEP and BSP efficiently
Efficient Persist Barriers for Multicores

Arpit Joshi, Vijay Nagarajan, Marcelo Cintra, Stratis Viglas