CCHL: Compression-Consolidation Hardware Logging for Efficient Failure-Atomic Persistent Memory Updates

Xueliang Wei, Dan Feng, Wei Tong, Jingning Liu, Chengning Wang, Liuqing Ye

Huazhong University of Science and Technology
Persistent Memory

- Provide data persistence at main-memory level
- Reduce persistence overhead compared with using traditional storage devices
Failure-Atomic Updates

- **Example**: Insert a node into a linked list in persistent memory

![Diagram]

Failure-Atomic Updates:
Persist a group of writes in an all or nothing manner in the presence of system failures
Durable Transactions

- **Example**: A durable transaction with write-ahead logging

```plaintext
C
  ↓ Insert
... A  B  ...

**Execute**

```

```plaintext
Tx_Begin
  Compute new data
  Log A
  Log C
  CLWB
  MFENCE
  St C, c1
  St A, a1
  CLWB
  MFENCE
Tx_End
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
```

```
**Full/Delayed Transaction Durability**

- **Example:** Fully/Delayed durable transactions with redo logging

**Full durability:**
The transaction is persisted during commit

**Delayed durability:**
The transaction can be persisted after commit
Software/Hardware Logging

- **Example**: Durable transactions with software/hardware redo logging

### Software Logging

Software performs log writes on the critical path of transaction execution, causing up to 70% performance degradation [ATOM, HPCA'17]

- **Tx_Begin**
  - Compute
  - Log a1
  - CLWB
  - MFENCE
  - St C, c1
  - St A, a1

- **Tx_End**
  - Persist Data

### Hardware Logging

Hardware performs log writes, asynchronous to volatile execution

- **Tx_Begin**
  - Compute
  - St C, c1
  - St A, a1

- **Tx_End**
  - Persist Data
Overview

• **Motivation:** *Many log writes are still executed in the critical path* in hardware logging, particularly for the multi-core systems with many threads.

• **Our Approach:** Eliminate unnecessary log writes and enable delayed transaction durability.
  - **Intra-Tx Log Compression**
    - **Observation 1:** 29.5% of data updated in transactions are dirty.
    - Avoid redundant log writes by *logging only dirty data*.
  - **Inter-Tx Log Consolidation**
    - **Observation 2:** 53.4% of data are updated by two close transactions (distance < 4).
    - Avoid redundant log writes by *combining successive transactions* when they update the same data.

• **Evaluation:** Improve performance by 47.4%, reduce PM write traffic by 36.1%, and reduces memory dynamic energy by 18.7%.
Outline

• Motivation
  • CCHL: Compression-Consolidation Hardware Logging
    • Intra-Tx Log Compression
    • Inter-Tx Log Consolidation
  • Evaluation
  • Conclusion
Execution Flow with Hardware Logging

- **Example**: Transaction execution flow with hardware redo logging
**Example:** Transaction execution flow with hardware redo logging

1. **Compute**

- **Cores:** a1, c1
- **Caches:** a0, c0
- **Memory Controller:** Write Queue
- **PM:** a0, b0, c0
- **Log Region**

**Execution Flow**

- **Tx_Begin**
- **Compute**
  - St C, c1
  - St A, a1
- **Tx_End**
Execution Flow with Hardware Logging

- **Example**: Transaction execution flow with hardware redo logging

- **Tx_Begin**
  - Compute
  - St C, c1
  - St A, a1

- **Tx_End**

- **Cores**
  - a1
  - c1

- **Caches**
  - a0
  - Log(C)

- **Memory Controller**
  - Write Queue

- **PM**
  - a0
  - b0
  - c0

- **Home Region**
- **Log Region**

- **1 Compute**
- **2 Write Data**
- **3 Write Log**
  - Log Buffer
Example: Transaction execution flow with hardware redo logging

- **Tx_Begin**
  - Compute
  - St C, c1
  - St A, a1
- **Tx_End**
Execution Flow with Hardware Logging

- **Example**: Transaction execution flow with hardware redo logging

```
Tx_Begin
  Compute
  St C, c1
  St A, a1
Tx_End
```

1. **Compute**
2. **Write Data**
3. **Write Log**
4. **Persist Log**
• **Example**: Transaction execution flow with hardware redo logging

**Tx_Begin**
- Compute
- St C, c1
- St A, a1

**Tx_End**

1. **Compute**
2. **Write Data**
3. **Write Log**
   - Log Buffer
4. **Persist Data**
5. **Persist Log**
Analysis of Hardware Logging Overhead

- Some log writes are still executed in the critical path
- **Example 1**: Evict a cache line when the write queue is full

![Diagram showing the process of hardware logging over different components: Cores, Caches, Memory Controller, and PM (Physical Memory). The diagram includes paths for log writes and the critical path.]
Analysis of Hardware Logging Overhead

- Some log writes are still executed in the critical path
- **Example 2**: Commit a transaction when some log entries are buffered
• Hardware logging overhead increases as the thread number increases
• The percentage of log writes increases as the thread number increases
Outline

• Motivation
• CCHL: Compression-Consolidation Hardware Logging
  • Intra-Tx Log Compression
  • Inter-Tx Log Consolidation
• Evaluation
• Conclusion
**Dirty data**: The data of which values are modified by transactions

**Observation 1**: Only 29.5% bytes among all the updated words are dirty
Intra-Tx Log Compression

- Only the log data for dirty data are essential for recovery
Intra-Tx Log Compression

• **Key idea**: Avoid redundant log writes by logging only dirty bytes
  • A \((p,q)\) dirty flag is added in each log entry to track the dirtiness of data
    • \((p,q)\) means the dirtiness of every \(q\)-byte data is tracked with \(p\) flag bits

```
<table>
<thead>
<tr>
<th>a0</th>
<th>0 0 0 0 0 0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>0 1 0 2 0 0 3 0</td>
</tr>
<tr>
<td>Log(A)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

Less clean data costs

Less dirty flag costs
Intra-Tx Log Compression

• How does intra-tx log compression reduce log writes?

8 log writes

MD

MD

MD

MD

LD 0

LD 1

LD 2

LD 3

Intra-tx log compression

MD: Metadata
Flag: Dirty flag
LD: Log data
CLD: Compressed log data

5 log writes

MD Flag

MD Flag

MD Flag

MD Flag

CLD 0

CLD 1

CLD 2

CLD 3

Log Packing
[Jeong+ MICRO’18]

Reduce 5 log writes

3 log writes

CLD 0

CLD 1

CLD 2

CLD 3
Intra-Tx Log Compression

• Implementation

Get dirty flag by comparing the old and the new value
Outline

• Motivation
• CCHL: Compression-Consolidation Hardware Logging
  • Intra-Tx Log Compression
  • Inter-Tx Log Consolidation
• Evaluation
• Conclusion
• **Transaction distance**: The number of transactions between two transactions that update the same words

• **Observation 2**: 53.4% of the updated words are written by two transactions of which the distance is less than 4
• Reduce log writes by avoiding writing unused log entries when several transactions update the same data
Inter-Tx Log Consolidation

- **Key idea**: Combine several successive transactions into a large one if they update the same data, and only log the newest values of the data.

```
<table>
<thead>
<tr>
<th>Tx_Begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>St A, a1</td>
</tr>
<tr>
<td>St B, b1</td>
</tr>
<tr>
<td>Tx_End</td>
</tr>
</tbody>
</table>
```

```
<table>
<thead>
<tr>
<th>Tx_Begin</th>
</tr>
</thead>
<tbody>
<tr>
<td>St C, c2</td>
</tr>
<tr>
<td>St B, b2</td>
</tr>
<tr>
<td>Tx_End</td>
</tr>
</tbody>
</table>
```

```
Log(A) = a1
Log(B) = b1
Log(C) = c2
Log(B) = b2
```

**Inter-tx log consolidation**

```
Log(A) = a1
Log(C) = c2
Log(B) = b2
```

- **4 log entries**

- **3 log entries**
Inter-Tx Log Consolidation

- Limitation

**Without inter-tx log consolidation:**

- **Tx1**
  - Tx_Begin
  - St A, a1
  - St B, b1
  - Tx_End
- **Tx2**
  - Tx_Begin
  - St C, c2
  - St B, b2
  - Tx_End

Failure happens

PM

<table>
<thead>
<tr>
<th>Home Region</th>
<th>Log Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>a1</td>
<td>Log(A) = a1</td>
</tr>
<tr>
<td>b1</td>
<td>Log(B) = b1</td>
</tr>
<tr>
<td>c0</td>
<td>Log(C) = c2</td>
</tr>
</tbody>
</table>

**Recovery**

**With inter-tx log consolidation:** *The updates of both Tx1 and Tx2 are lost*

PM

<table>
<thead>
<tr>
<th>Home Region</th>
<th>Log Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>a0</td>
<td>Log(A) = a1</td>
</tr>
<tr>
<td>b0</td>
<td>Log(C) = c2</td>
</tr>
<tr>
<td>c0</td>
<td></td>
</tr>
</tbody>
</table>
Inter-Tx Log Consolidation

- Implementation

**DRAM Cache**

- **Tx1**
  - **Tx_Begin**
  - St A, a1
  - St B, b1
  - **Tx_End**
  - **Tx2**
  - **Tx_Begin**
  - St C, c2
  - St B, b2
  - **Tx_End**

**Log(A)** = a1
**Log(B)** = b1
**Log(C)** = c2

**PM**

- **a0**
- **b0**
- **c0**

**Home Region**

**Log Region**

**Log(B)** = b2
Outline

• Motivation
• CCHL: Compression-Consolidation Hardware Logging
  • Intra-Tx Log Compression
  • Inter-Tx Log Consolidation
• Evaluation
• Conclusion
Experimental Setup

- **Gem5 simulator configuration**
  - Eight-core processor with private L1 and L2 caches and a shared L3 cache
  - 16-entry log buffer and (1,1) dirty flag
  - Memory parameters from [Ren+ MICRO’15, Lee+ ISCA’09, Ogleari+ HPCA’18]

- **Workloads**
  - Micro-benchmarks: Btree, Hash, Queue, RBTree, SDG, SPS
  - Macro-benchmarks [Nalli+ ASPLOS’17]: Echo, YCSB, TPCC

- **Evaluated designs**
  - **ATOM**: Hardware undo logging with full durability
  - **FWB**: Hardware undo+redo logging with full durability
  - **ReDU**: Hardware redo logging with full durability
  - **CCHL-fd**: Proposed design with full durability and intra-tx log compression
  - **CCHL-dwN**: Proposed design with delayed durability and both log optimizations
• CCHL-fd outperforms ReDU by 47.8% for the small dataset sizes
• CCHL-fd outperforms ReDU by 47.0% for the large dataset sizes
### Write Traffic and Energy Consumption

<table>
<thead>
<tr>
<th>Dataset size</th>
<th>ATOM</th>
<th>FWB</th>
<th>ReDU</th>
<th>CCHL-fd</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Normalized PM write traffic</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>1.00</td>
<td>1.49</td>
<td>0.99</td>
<td>0.64</td>
</tr>
<tr>
<td>Large</td>
<td>1.00</td>
<td>1.45</td>
<td>1.00</td>
<td>0.63</td>
</tr>
<tr>
<td><strong>Normalized memory dynamic energy</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Small</td>
<td>1.00</td>
<td>1.63</td>
<td>1.00</td>
<td>0.79</td>
</tr>
<tr>
<td>Large</td>
<td>1.00</td>
<td>1.53</td>
<td>0.80</td>
<td>0.67</td>
</tr>
</tbody>
</table>

- CCHL-fd significantly reduces both PM write traffic and memory dynamic energy
# Efficiency of Intra-Tx Log Compression

<table>
<thead>
<tr>
<th>Type</th>
<th>ATOM+C</th>
<th>FWB+C</th>
<th>ReDU+C</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transaction throughput improvement</td>
<td>23.3%</td>
<td>33.1%</td>
<td>29.5%</td>
</tr>
<tr>
<td>PM write traffic reduction</td>
<td>34.4%</td>
<td>43.0%</td>
<td>34.4%</td>
</tr>
<tr>
<td>Memory dynamic energy reduction</td>
<td>14.7%</td>
<td>22.0%</td>
<td>16.0%</td>
</tr>
</tbody>
</table>

- “+C” represents the corresponding design with intra-tx log compression
- All the three designs benefit from intra-tx log compression
Efficiency of Inter-Tx Log Consolidation

- CCHL-dwN (N=2,4,6,8) improves transaction throughput on all the workloads through delayed durability and inter-tx log consolidation
Outline

• Motivation
• CCHL: Compression-Consolidation Hardware Logging
  • Intra-Tx Log Compression
  • Inter-Tx Log Consolidation
• Evaluation
• Conclusion
Conclusion

• **Motivation**: Many log writes are still executed in the critical path in hardware logging, particularly for the multi-core systems with many threads.

• **Key Idea**: Eliminate unnecessary log writes and enable delayed transaction durability.
  
  • **Intra-Tx Log Compression**
    
    • **Observation 1**: 29.5% of data updated in transactions are dirty.
    
    • Avoid redundant log writes by *logging only dirty data*.

  • **Inter-Tx Log Consolidation**
    
    • **Observation 2**: 53.4% of data are updated by two close transactions (distance < 4).
    
    • Avoid redundant log writes by *combining successive transactions* when they update the same data.

• **Evaluation**: Improve performance by 47.4%, reduce PM write traffic by 36.1%, and reduces memory dynamic energy by 18.7%.
Thank you!

CCHL: Compression-Consolidation Hardware Logging for Efficient Failure-Atomic Persistent Memory Updates

Xueliang Wei, Dan Feng, Wei Tong, Jingning Liu, Chengning Wang, Liuqing Ye

Huazhong University of Science and Technology