Using Software Logging To Support Multi-Version Buffering in Thread-Level Speculation

María Jesús Garzarán, M. Prvulovic, V. Viñals, J. M. Llabería*, L. Rauchwergerψ, and J. Torrellas

U. of Illinois at Urbana-Champaign  §U. of Zaragoza, Spain

* U.P.C., Spain  ψ Texas A&M University
Thread-Level Speculation (TLS)

- Execute potentially-dependent tasks in parallel
  - Assume no cross-task dependence will be violated
  - Track memory accesses; buffer unsafe state
  - Detect any dependence violation
  - Squash offending tasks, repair polluted state, restart tasks

Example

```plaintext
time

for (i=0; i<n; i++) {
    ... = A[B[i]] ...  
    A[C[i]] = ...
}
```

Task J

- Task J
- Task J+1
- Task J+2

RAW

A[5] = ...
A[2] = ...
A[6] = ...

... = A[4] ...
... = A[2] ...
... = A[5] ...
... = A[6] ...
State Buffering in TLS

- Speculative tasks generate speculative memory state
- Must buffer and manage this speculative state
Approaches to Buffering [HPCA-03]

- **Architectural Main Memory (AMM):**
  - Speculative task state kept in caches
  - Main memory only keeps safe program state
  - Cached state merges with memory when a task commits

- **Future Main Memory (FMM):**
  - Speculative task state can merge with memory at any time
  - Main memory keeps the future state (unsafe)
  - Previous state is saved into *Undo Log*
Our Contributions

- Software-only design for undo-log system in FMM
- Simplify the hardware implementation
- Very cost-effective approach
  - On average, it only introduces a 10% slow-down vs hardware-only
Roadmap

- Taxonomy of Buffering: AMM vs FMM
- Hardware Support
- Software Support
- Evaluation
- Conclusions
Task Execution under TLS
**Architectural Main Memory** [HPCA03]

- **Main memory keeps architectural or safe state**
- **Caches keep speculative state**
Future Main Memory [HPCA03]

- Main memory keeps future state
- Logs keep previous state
Future Main Memory

<table>
<thead>
<tr>
<th>value</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task i writes 2 to 0x400</td>
<td></td>
</tr>
<tr>
<td>Task i+j writes 10 to 0x400</td>
<td></td>
</tr>
</tbody>
</table>

Architectural MM

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0x400</td>
<td>2</td>
</tr>
<tr>
<td>i+j</td>
<td>0x400</td>
<td>10</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0x400</td>
<td>2</td>
</tr>
</tbody>
</table>

Future MM

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0x400</td>
<td>2</td>
</tr>
</tbody>
</table>

Cache

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>i+j</td>
<td>0x400</td>
<td>10</td>
</tr>
</tbody>
</table>
Future Main Memory

<table>
<thead>
<tr>
<th>Task ID</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>0x400</td>
<td>2</td>
</tr>
<tr>
<td>i+j</td>
<td>0x400</td>
<td>10</td>
</tr>
</tbody>
</table>

Task i writes 2 to 0x400
Task i+j writes 10 to 0x400

Architectural MM

Future MM

Faster commit but slower version recovery
Need log
Roadmap

- Taxonomy of Buffering
- **Hardware Support**
- Software Support
- Evaluation
- Conclusions
Hardware Supports

- Existing TLS protocol
- Add hooks to support Software Logging
TLS protocol [Zhang99]
Hooks to Support Software Logging

1) Make Task ID pages visible to the software

Processor

Cache

Network

Task IDs

Local Memory

Software Log

<table>
<thead>
<tr>
<th>Producer Task ID</th>
<th>Data</th>
</tr>
</thead>
</table>

MaxR | MaxW | X

Producer

Task ID Log
Accessing Task IDs in Software

- Fixed offset between mapping of data pages and corresponding Task ID pages

- Use two different loads:
  - ld variable
  - ld_TID variable
Accessing Data

Virtual page  Physical DataPage

TLB

ld variable

To cache (data access)
Accessing Task IDs

TLB

Virtual page   Physical DataPage

ld_TID variable

Fixed offset

To cache (task ID access)
Caching Task IDs

- Bring Task ID to cache with `ld_TID` instruction as regular data.

- Task IDs can be reused from the cache.

- Task IDs in memory are updated in hardware by the TLS protocol.

- To keep cached Task IDs up-to-date in software: `st_TID` instruction.
Roadmap

- Taxonomy of Buffering
- Hardware Support
- **Software Support**
- Evaluation
- Conclusions
A compiler instruments the application:
- **Insert entry in log**: before a store operation, add instructions to save previous value, address and Task ID in log
- **Recycle log entries**: free up the log created when a task commits

**Interrupt handler:**
- **Recovery**:
  - In case of a o-o-o RAW and squash
  - Undo the modifications using data from log
Software Data Structures

- Logs are allocated locally before speculation starts

<table>
<thead>
<tr>
<th>Valid</th>
<th>Task ID</th>
<th>End</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>j</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

![Task Pointer Table](image)

![Log Buffer](image)

22
Instructions to insert entry in log

Logging instr

- `addu r4, r3, offset` ; address of the variable
- `sw r4, 0(r2)` ; store in the log
- `ld_TID r4, offset(r3)` ; load Task ID
- `sw r4, 4(r2)` ; store in the log
- `lw r4, offset(r3)` ; load value of variable
- `sw r4, 8(r2)` ; store in the log
- `addu r2, r2, log_record_size`
- `st_TID r5, offset(r3)` ; store Task ID
- `sw r5, offset(r3)`
Reducing Overheads

- Only **first-stores** in the task need to create a log entry
- Solution: At run-time, check if store is first-store
- Use cached Task ID to filter:
  - If (Cached Task ID == Current Task ID)
    Skip Logging
  - Else
    Log entry
Filtering First Speculative Store

```
ld_TID r6, offset (r3); load Task ID
beq r6, r5, no_insert; first store?
addu r4, r3, offset; insert as usual
sw r4, 0(r2)

 Logging

 instr

 no_insert:

 sw r5, offset (r3)
```
Roadmap

- Taxonomy of Buffering
- Hardware Support
- Software Support
- Evaluation
- Conclusions
Evaluation Environment

- Execution-driven simulator
  - Multiprocessor with 16 processors

- 4 issue o-o-o superscalar processor + 2 levels of cache

- Compare against
  - FMM with Hardware Logging [Zhang99]
  - Advanced AMM system [HPCA03]
Applications

- Numerical applications:
  - Apsi (Specfp2000)
  - Dsmc3d and Euler (HPF-2)
  - P3m (NCSA)
  - Tree (Univ. of Hawaii)
  - Track (Perfect)

The non-analyzable loops account on average for 61% of the serial execution time.

- Non-analyzable loops are identified with the Polaris parallelizing compiler
- Speed-ups shown for the non-analyzable loops only
Execution Time Comparison

- On average, FMM Sw only introduces a 10% slow-down over FMM Hw
- On average, FMM Sw is similar to the advanced AMM
First Store Filtering

- Significant impact in the execution time in Apsi
- Since filtering does not hurt, we recommend using it
Other Results in Paper

- Studied design tradeoffs:
  - Log accesses bypass/do not bypass L1 cache
  - Log space is / is not recycled

Do not affect the performance of FMM.Sw when filtering is used
Conclusions

- FMM.Sw is a cost-effective solution:
  - Simplified design relative to FMM.Hw
  - Introduce low execution overhead (10% over FMM.Hw)

- Filtering first stores is beneficial
Using Software Logging To Support Multi-Version Buffering in Thread-Level Speculation

**María Jesús Garzarán**, M. Prvulovic, V. Viñals,§ J. M. Llabería*, L. Rauchwergerψ, and J. Torrellas

U. of Illinois at Urbana-Champaign  §U. of Zaragoza, Spain

* U.P.C., Spain  ψ Texas A&M University
FMM.Sw versus Advanced AMM

- **Advanced AMM**
  - Needs Version Combining Logic
    - Collect all the versions that are committed
    - Select the youngest one and invalidate the others

- **FMM.Sw**
  - Needs Task ID in main memory
  - Comparator to pick up the youngest version
Problem: Address task IDs in software

TLB

Non-existent Physical address

Cache
data & ld/st bits

Cache tagged with Non-existent physical addr

Local Memory
task ID

Add a fixed offset
Problem: Address task IDs in software

TLB

Non-existent Physical address

Cache

Data & ld/st bits

Local Memory

Task ID

NIC

Physical address

local

shared

Network

Shared Data
Problem: Address time stamp in software

- **OS**
  - Allocates a page of task IDs
  - Map the virtual address to a non existent physical page

![Diagram showing memory and network components](image)
Speculative protocol

- **Cache**: load and store bits per word in cache
- **Local Memory**: task ID per word
- **ISA**: new ld/st instructions
**Problem: Address task IDs in Software**

- The task ID is not mapped in virtual space
- How to make visible the task ID to the sw?

Logging inst

\[
\begin{cases}
\text{load r3, addr_task ID} \\
\text{sw r5, offset(r3)}
\end{cases}
\]

<table>
<thead>
<tr>
<th>Undo Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vaddr</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td>---</td>
</tr>
</tbody>
</table>
Problem: Address task IDs in Software

- The task ID is not mapped in virtual space
- How to make visible the task ID to the sw?
  - Use special instruction lh_TID

Logging

\[
\begin{align*}
\text{lh}_\text{TID} & \quad r3, \quad \text{offset}(r3) \\
\text{sw} & \quad r5, \quad \text{offset}(r3)
\end{align*}
\]

Undo Log

<table>
<thead>
<tr>
<th>Vaddr</th>
<th>task ID</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Accessing Task IDs in Software

- Option 2) Map to a non-existent physical address

Virtual page | Physical Non existent DataPage

\[ \text{lh}_\text{TID} \ r3, \ \text{offset}(r3) \]

Fixed offset

To cache (task ID access)
Reducing Overheads

- Only **first-stores** in the task need to create a log entry
- Solution: At run-time, check if store is first-store
- Use cached Task ID to filter:
  - If

![Diagram showing first stores, speculative, and non-speculative stores]

- Instrumented
  - First stores
    - Speculative
    - Non speculative
  - Others
Filtering first speculative store

- Using extended loads

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xlw r6, r1, offset (r3)</td>
<td>Store bit goes to r6</td>
</tr>
<tr>
<td>bgtz r6, no_insert</td>
<td>first store?</td>
</tr>
<tr>
<td>addu r4, r3, offset</td>
<td>insert as usual</td>
</tr>
<tr>
<td>sw r4, 0(r2)</td>
<td></td>
</tr>
<tr>
<td>lh_ts r4, offset(r3)</td>
<td></td>
</tr>
</tbody>
</table>

Logging instr:

- addu r2, r2, log_record_size
- sh_TID r5, offset (r3)

no_insert:

- sw r5, offset(r3)
Software handlers

- **Recovery**: Out-of order RAW
  - Undo the modifications using data from log

- **Retrieval**: Some in-order RAWs
  - The exposed load needs dig version from log
Accessing Task IDs in Software

- Naïve solution: Add extra field in TLB

```
Virtual    Physical    Physical
Page       Data       Task ID
            Page       Page

Id var
```

To cache
(data access)
Accessing Task IDs in Software

- Naïve solution: Add extra field in TLB

Virtual Page | Physical Data Page | Physical Task ID Page

ld_TID var To cache (task ID access)
Accessing Task IDs in Software

- Fixed offset between mapping of data pages and corresponding Task ID pages

- No TLB modifications

- In `ld_TID` instruction, the hardware subtracts the offset to obtain the Physical Task ID page