Tradeoffs in Buffering Memory State for Thread-Level Speculation in Multiprocessors

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

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

* U.P.C. , Spain ψ Texas A&M University
Motivation: Speculative State Management under TLS

- Thread-Level Speculation (TLS) extracts parallelism from hard-to-analyze applications
  - Pointers, arrays with subscripted subscripts, …

- Speculative tasks generate speculative memory state

- Must buffer and manage this speculative state
Motivation: Speculative State Management under TLS

- Speculative memory state buffering varies across TLS schemes:
  - Where is the speculative state buffered?
  - How many speculative tasks and versions are supported?
  - How is the state merged with main memory?

→ Must understand the performance/complexity tradeoffs
Our Contributions

- New taxonomy of buffering schemes under TLS
  - Identifies major axes in the design space

- Performance/Complexity tradeoff analysis of schemes
  - Performance benefits
  - Design complexity
  - Application characteristics

- Performance comparison of schemes with a unified framework:
  - Same speculation protocol
  - Same architectural parameters
  - Same applications
Roadmap

- Introduction to Thread-Level Speculation (TLS)
- Taxonomy of Buffering
- Tradeoffs Analysis
- Evaluation
- Conclusions
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**

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

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

```
... = A[4] ...
... = A[2] ...
... = A[5] ...
... = A[6] ...
```

RAW
Task Execution under TLS

Processor

\[ \text{time} \]

1

Non Spec

Commit
Token

task 1

task 4

2

Spec

task 2

3

Spec

task 3
Task Execution under TLS
Task Execution under TLS

Processor

1  Non Spec
   task 1
   task 4

2  Non Spec
   task 2
   task 5

Commit Token

3  Spec
   task 3
Task Execution under TLS

Processor

1. Non Spec
   - task 1
   - task 4

2. Non Spec
   - task 2
   - Commit Token
   - task 5

3. Non Spec
   - task 3
Challenges in Buffering Speculative State

- State needs to be buffered until a task becomes non-speculative
- State must be merged in order

Tasks

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caches</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
</tbody>
</table>

Non spec

Main memory
Roadmap

- Introduction to Thread-Level Speculation
- Taxonomy of buffering
- Tradeoffs Analysis
- Evaluation
- Conclusions
Taxonomy of Buffering

Merging of task state

Separation of task state
Separation of Task State
Taxonomy of Buffering

- Merging of task state
- Separation of task state
- Multiple Spec Tasks
- Single Spec Task Per Proc
Merging of Task State: **Architectural** Main Memory

- Main memory keeps architectural or safe state
- Caches keep speculative state
Merging of Task State: **Future** Main Memory

- Main memory keeps future state
- Logs keep previous state
Taxonomy of Buffering

Merging of task state

Separation of task state

Multiple Spec Tasks

Single Spec Task Per Proc

Architectural Main Memory

Future Main Memory
Merging with Architectural Main Memory: **EAGER**

- State is merged with main memory at commit time
Merging to Architectural Main Memory: **LAZY**

State is merged with main memory *at or after* the commit.
Taxonomy of Buffering

Merging of task state

Separation of task state

Multiple Spec Tasks

Single Spec Task Per Proc

Architectural Main Memory

Future Main Memory

Eager

Lazy
Mapping Existing Schemes

- Merging of task state
- Separation of task state

Architectural Main Memory
- Eager
- Lazy

Future Main Memory
- Hydra
- Steffan Prvulovic
- Zhang

Hydra
- Multiscalar (H-ARB)
- Super-threaded

Steffan Prvulovic
- Multiscalar (SVC)

Zhang
- SUDS

Multiple Spec Tasks

Single Spec Task Per Proc
Roadmap

- Introduction to Thread-Level Speculation
- Taxonomy of Buffering
- Tradeoffs Analysis
- Evaluation
- Conclusions
Comparing Schemes

Merging of task state
Separation of task state

Architectural Main Memory
Eager | Lazy

Future Main Memory

Multiple Spec Tasks
Single Spec Task Per Proc
Multiple Spec Tasks / Proc

Single Task per Processor

- Perf: Tolerate load imbalance
- Cost: Need Task ID in cache lines & advanced comparison logic in caches

Multiple Tasks per Processor

- Ei: Execution of Task i
- Ci: Commit of Task i
Comparing Schemes

- Merging of task state
- Separation of task state

<table>
<thead>
<tr>
<th>Architectural Main Memory</th>
<th>Future Main Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eager</td>
<td>Lazy</td>
</tr>
</tbody>
</table>

- Multiple Spec Tasks
- Single Spec Task Per Proc
Two wavefronts: execution + commit

Processor 0 1 2 3

Execution Wavefront

E0 E1 E2 E3
E4 E5 E6 E7
E8 E9 E10 E11
Two wavefronts: execution + commit

<table>
<thead>
<tr>
<th>Processor</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E1</td>
<td>C0</td>
<td>C1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>E2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Commit Wavefront
Eager vs Lazy Merging in Architectural MM

Commit wavefront appears in critical path when:

\[
C_i \times Nprocs > E_i
\]
Eager vs Lazy Merging in Architectural MM

Eager

Proc 0 1 2 3

\[ E_0 \quad E_1 \quad E_2 \quad E_3 \]

Proc 0 1 2 3

\[ C_0 \quad C_1 \quad C_2 \quad C_3 \]

\[ C_4 \quad C_5 \quad C_6 \quad C_7 \]

Lazy

Removes the commit wavefront from the critical path
Eager vs Lazy Merging in Architectural MM

**Eager**

<table>
<thead>
<tr>
<th>Proc</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>E0</td>
<td>E1</td>
<td>E2</td>
</tr>
<tr>
<td>C1</td>
<td></td>
<td></td>
<td>Stall</td>
</tr>
<tr>
<td>C2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Lazy**

<table>
<thead>
<tr>
<th>Proc</th>
<th>0</th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>E0</td>
<td>E1</td>
<td>E2</td>
<td></td>
</tr>
<tr>
<td>E3</td>
<td>C0</td>
<td>C1</td>
<td>C2</td>
</tr>
</tbody>
</table>

**Perf**
- Remove commit wavefront from critical path

**Cost**
- Need Task ID in cache lines
- Need version combining logic
Comparing Schemes

Merging of task state

Separation of task state

Multiple Spec Tasks

Single Spec Task Per Proc

Architectural Main Memory

Future Main Memory

Eager Lazy
Future Main Memory

value address

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

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>

Cache

Perf: Faster commit but slower version recovery
Cost: Need log

Future MM

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

Cache

Producer

Task ID

Overwriting

Log (in cache or memory)

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

Cost: Need Task ID tags in main memory
Recap

Merging of task state

Separation of task state

Multiple Spec Tasks

Single Spec Task Per Proc

Architectural Main Memory

Future Main Memory

Version Combining Logic

Task ID Comparison Logic

Log Task ID in Main Memory

Task ID Version Combining Logic
Roadmap

- Introduction to Thread-Level Speculation
- Taxonomy of Buffering
- Tradeoffs Analysis
- Evaluation
- Conclusions
Evaluation Environment

- Execution-driven simulator
  - CC-NUMA multiprocessor with 16 processors
  - CMP (Chip multiprocessor) with 8 processors

- Out-of order superscalar processor + 2 levels of cache
  - Issues 4 instructions per cycle
Applications

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

  The non-analyzable loops account on average for 59% of the serial execution time.

- Non-analyzable loops are identified with the Polaris parallelizing compiler.

- Speed-ups shown for the non-analyzable loops only.
Supporting Multiple Tasks

- Merging of task state
- Separation of task state
- Architectural Main Memory
  - Eager
  - Lazy
- Future Main Memory
- Multiple Spec Tasks
- Single Spec Task Per Proc
CC-NUMA: Supporting Multiple Tasks (Eager)

<table>
<thead>
<tr>
<th>Application</th>
<th>Single Task</th>
<th>Multiple Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3m</td>
<td>0.7</td>
<td>7.9</td>
</tr>
<tr>
<td>Tree</td>
<td>11.9</td>
<td>14.1</td>
</tr>
<tr>
<td>Bdna</td>
<td>4.2</td>
<td>3.4</td>
</tr>
<tr>
<td>Apsi</td>
<td>1.9</td>
<td>7.9</td>
</tr>
<tr>
<td>Track</td>
<td>1.8</td>
<td>7.9</td>
</tr>
<tr>
<td>Dsmc3d</td>
<td>6.2</td>
<td>7.5</td>
</tr>
<tr>
<td>Euler</td>
<td>0.5</td>
<td>0.6</td>
</tr>
</tbody>
</table>

Average Speedup:
- Single Task: 0.6
- Multiple Tasks: 1.4

Diagram showing normalized execution time comparison between single and multiple tasks for various applications.
Supporting Multiple Tasks (Eager)

Single Task per Processor

Multiple Tasks per Processor

Ei: Execution of Task i
Ci: Commit of Task i
CC-NUMA: Supporting Multiple Tasks (Eager)
Supporting Multiple Tasks (Eager)
CC-NUMA: Supporting Multiple Tasks (Eager)

- Apps run 32% faster with Multiple Tasks
Adding laziness to Single Task

Merging of task state

Separation of task state

Multiple Spec Tasks

Single Spec Task Per Proc

Architectural Main Memory Eager

Future Main Memory Lazy

Single Task Per Proc

Multiple Tasks
CC-NUMA: Single Task Eager & Lazy Merging

<table>
<thead>
<tr>
<th></th>
<th>P3m</th>
<th>Tree</th>
<th>Bdna</th>
<th>Apsi</th>
<th>Track</th>
<th>Dsmc3d</th>
<th>Euler</th>
<th>AVERAGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Execution Time</td>
<td>0.7</td>
<td>11.9</td>
<td>12.3</td>
<td>4.2</td>
<td>1.9</td>
<td>6.2</td>
<td>3.6</td>
<td>0.5</td>
</tr>
</tbody>
</table>

Normalized Execution Time

![Chart showing normalized execution time for different tasks with eager and lazy merging]
Single Task Eager & Lazy Merging

Eager

Proc 0 1 2
E0 E1 E2
C0 C1 C2
Stall

Lazy

Proc 0 1 2
E0 E1 E2
C0 C1 C2

Single Task Per Processor
CC-NUMA: Single Task Eager & Lazy Merging

Laziness speeds up the applications 30%
Multiple Tasks & Laziness

- Architectural Main Memory
  - Eager
  - Lazy
- Future Main Memory
- Merging of task state
- Separation of task state
- Multiple Spec Tasks
- Single Spec Task Per Proc
CC-NUMA: Multiple Tasks & Laziness

Gains from Multiple Tasks and Laziness are fairly orthogonal (48% speedup)
Architectural and Future MM

Merging of task state
Separation of task state

Architectural Main Memory
Eager

Future Main Memory
Lazy

Multiple Spec Tasks
Single Spec Task Per Proc
CC-NUMA: Architectural and Future Main Memory

- Future and Lazy Architectural MM have similar performance
- Future MM is better if cache pressure (P3m)
- Future MM may suffer if frequent squashes (Euler)
Summary

Merging of task state

Separation of task state

Multiple Spec Tasks

Single Spec Task Per Proc

Architectural Main Memory
Eager
Lazy

Future Main Memory

32% 48%
same

30%
Conclusions

1. Adds modest complexity

2. Additional performance gains
   Targets an orthogonal problem

3. Similar performance
   Makes the system more robust
   Can suffer if frequent dependence violations

Merging of task state
Separation of task state

Multiple Spec Tasks
Single Spec Task

Architectural Main Memory
Eager Lazy

Future Main Memory

1 2 3
Tradeoffs in Buffering Memory State for Thread-Level Speculation in Multiprocessors

María Jesús Garzarán, M. Prvulovic§, J. M. Llaberia*, V. Viñals, L. Rauchwerger¶, and J. Torrellas§

garzaran@cs.uiuc.edu
http://iacoma.cs.uiuc.edu/
http://www.cps.unizar.es/gaz
App characteristics that limit performance

- Merging of task state
- Separation of task state
- Single Spec Task Per Proc
- Multiple Spec Tasks
- Architectural Main Memory: Eager
- Future Main Memory: Lazy
- Large Working Set
- Imbalance
- Large Commit/Execution
- Frequent dependence violations
Summary

- Merging of task state
- Separation of task state
  - Architectural Main Memory
    - Eager
    - Lazy
  - Future Main Memory
  - Same
- Multiple Spec Tasks
  - 32% 23%
- Single Spec Task Per Proc
  - 30% 9%

CC-NUMA  CMP
CC-NUMA vs CMP: Architectural MM

- Less differences in CMP
Supporting Multiple Tasks (Eager)

Single Task

Proc 0 1 2

E0 E1 E2

C0 C1 C2

Multiple Tasks

Proc 0 1 2 3

E0 E1 E2 E3

E4

C0 C1 C2

C3 C4 C5 C6 C7
Task Execution under TLS

Processor

time

1

Non Spec

Commit Token

2

NSpec

Commit Token

3

NSpec

Commit Token

task 1

task 2

task 3

task 4

task 5

task 6
Task Execution under TLS

Processor

1. Non Spec
   - task 1
   - Commit Token
   - task 4

2. Non Spec
   - task 2
   - Commit Token
   - task 5

3. Non Spec
   - task 3
Eager vs Lazy Merging in Architectural MM

Multiple Tasks Per Processor

Commit wavefront appears in critical path when:

\[ C_i \times N\text{procs} > E_i \]
Thread-Level Speculation

Processor
time

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>task 1</td>
<td>task 2</td>
<td>task 3</td>
<td>task 4</td>
</tr>
<tr>
<td>2</td>
<td>CP = 1</td>
<td>CP = 2</td>
<td>CP = 3</td>
<td>CP = 4</td>
</tr>
<tr>
<td>3</td>
<td>task 5</td>
<td></td>
<td></td>
<td>task 6</td>
</tr>
</tbody>
</table>

- CP = 1
- CP = 2
- CP = 3
- CP = 4
Supporting Multiple Tasks

Single Task per Processor

0 1
E0 E1
C0 E2 C1 E3

Multiple Tasks per Processor

0 1
E0 E1
C0 C1

Ei: Execution of Task i
Ci: Commit of Task i
Challenges in Buffering Speculative State

- Several versions of the same variable in the system
- Main memory needs to be correctly updated
Conclusions

- Starting from SingleT Eager Architectural Main Memory
  - Adding **multiple T&V** is more cost-effective than lazy merging:
    - Higher performance
    - Lower implementation complexity
  - With multiple T&V, **lazy merging** further increases performance
    - Fairly orthogonal impact
  - Further supporting **Future MM**:
    - Modest increase in performance
    - Can suffer if frequent squashes
    - Adds complexity
Both Future and Lazy Architectural MM have similar perf
- Future MM is better if cache pressure (P3m)
- Future MM may suffer if frequent squashes (Euler)
Challenges in Speculative State Buffering - II

- Several versions of the same variable in the system
- Main memory needs to be updated in the correct order
Several versions of the same variable in the same cache
Separation of Task State

Tasks 3 4 5 6 7

Caches X T4

Main memory

Task ID T6 T4
## Application Characteristics*

*Executing in a DSM with 16 processors

<table>
<thead>
<tr>
<th>Application</th>
<th>Average # Speculative Tasks in the System</th>
<th>Average Written Footprint per Task</th>
</tr>
</thead>
<tbody>
<tr>
<td>P3m</td>
<td>800.0</td>
<td>Total (KB) 1.7 Priv. (%) 87.9</td>
</tr>
<tr>
<td>Tree</td>
<td>24.0</td>
<td>Total (KB) 0.9 Priv. (%) 99.5</td>
</tr>
<tr>
<td>Bdna</td>
<td>25.6</td>
<td>Total (KB) 23.7 Priv. (%) 99.4</td>
</tr>
<tr>
<td>Apsi</td>
<td>28.8</td>
<td>Total (KB) 20.0 Priv. (%) 60.0</td>
</tr>
<tr>
<td>Track</td>
<td>20.8</td>
<td>Total (KB) 2.3 Priv. (%) 0.6</td>
</tr>
<tr>
<td>Dsmc3d</td>
<td>17.6</td>
<td>Total (KB) 0.8 Priv. (%) 0.5</td>
</tr>
</tbody>
</table>

*Executing in a DSM with 16 processors
Goal

- Execute hard to analyze codes in parallel
  - Pointers
  - Indirectly-indexed structures
  - Possible interprocedural dependences
  - Input-dependent patterns

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

- Merging of task state
- Separation of task state
- Multiple Spec Tasks
- Single Spec Task Per Proc

Architectural Main Memory
- Eager
- Lazy

Future Main Memory

Imbalance

48% 25%
30% 9%