Software Logging under Speculative Parallelization

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

U. de Zaragoza
U. Politècnica de Catalunya

U. of Illinois
Texas A&M U.
Roadmap of the Talk

- Speculative tasks running in the same processor can create multiple versions of the same variable
  - Stall the processor or redesign the caches

- Alternative solution: Logs

- Contribution:

  Design, integration and evaluation of software logging on top of a speculation protocol
  - cheap, low overhead (10%)
Outline

- Speculative Parallelization
- Multiple Local Speculative Versions
- Software Logging
- Evaluation
- Conclusions
Speculative Parallelization

- Assume no dependences and execute tasks in parallel
- Track data accesses
- Detect violations
- Squash offending tasks and restart them

\[
\text{Do } I = 1 \text{ to } N \\
\ldots = A(L(I)) + \ldots \\
A(K(I)) = \ldots \\
\text{EndDo}
\]

\[
\text{Task } J \\
\ldots = A(4) + \ldots \\
A(5) = \ldots \\
\text{RAW}
\]

\[
\text{Task } J+1 \\
\ldots = A(2) + \ldots \\
A(2) = \ldots \\
\text{Task } J+2 \\
\ldots = A(5) + \ldots \\
A(6) = \ldots \\
\]
Speculative Parallelization

- Speculative tasks cannot displace speculative data
- State buffered until task becomes non-speculative
Several Tasks Share a Cache

- Processors must hold speculative state of several tasks
- Task-ID field to identify the owner  [Cintra00][Stefan00]
Outline

- Speculative Parallelization
- Multiple Local Speculative Versions
- Software Logging
- Evaluation
- Conclusions
Last and Non-Last Versions

- Speculative tasks in the same processor write the same memory address

Task 5:

store value1, 0x400 ← non-last version

Task 8:

store value2, 0x400 ← last version

....

load r4, 0x400 ← needs last version
Multiple Local Speculative Versions

- To avoid the stall of the processor:
  - Modify the cache
  - Use Logs
Modify the cache

- Cache keeps last and non-last versions (same Tag, but different task-ID)
  - complexity and extra comparisons
  - chances of displacement increase
  - equally hard access last versions than non-last versions

<table>
<thead>
<tr>
<th>Task-ID</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0x400</td>
<td>value1</td>
</tr>
<tr>
<td>8</td>
<td>0x400</td>
<td>value2</td>
</tr>
</tbody>
</table>
Logs

- Cache keeps last versions
- Logs hide away non-last versions

Cache

<table>
<thead>
<tr>
<th>Task-ID</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0x400</td>
<td>value2</td>
</tr>
</tbody>
</table>

Log

<table>
<thead>
<tr>
<th>Task-ID</th>
<th>Addr</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0x400</td>
<td>value1</td>
</tr>
</tbody>
</table>

Memory
Logs

- Collect the state that a task made stale
- Useful
  - Free up space when the task commits
  - To recover in case of squashes

![Diagram of logs and cache with task relationships]

- Task 10
- Task 8
- Task 5

Cache

Last versions

Undo Log

Non-last versions

Task 8

Task 10
Speculative protocol

- Speculative protocol using Hw logs was proposed:


- Use Sw Logs on top of a speculative protocol:
  - **Task-ID**: per memory word in the local memory
  - **ISA**: new ld/st instructions
Outline

- Speculative Parallelization
- Multiple Local Speculative Versions
- **Software Logging**
- Evaluation
- Conclusions
Software Logs

- A compiler instruments the application
  - Insert: extra instructions before store operations
  - Recycle: free up space when a task commits

- Interrupt handlers
  - Recovery: in case of a o-o-o RAW and squash
  - Retrieval: in-order RAW and the version in the log
Software Data Structures

- Logs are allocated locally before speculation starts

### Task Pointer Table

<table>
<thead>
<tr>
<th>Valid</th>
<th>Task ID</th>
<th>Ovflw</th>
<th>End</th>
<th>Next</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>i</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>j</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Log Buffer

- Owner
- Vaddr
- Task-ID
- Value

- Free Sector Stack
- Sector Stack
Instructions to insert in log

# Assembly Instruct

Check log overflow           1
Log. Vaddr                   =  addr of var        2
Log. OwnerTask_ID = current Task_ID  2
Log. Value                   =  value of var       2
Increment log pointer        1
Update Task_ID               1

Original store

-----------------------------
Total  9
Reducing unnecessary logging

- Create log entry: only 1st write to each variable
  - Non-spec vars: easy to identify
  - Spec vars: hard
    - Insert run-time check in all spec writes
    - If 1st, create log entry

==> Much reduced instrumentation overhead
Outline

- Speculative Parallelization
- Multiple Local Speculative Versions
- Software Logging
- Evaluation
- Conclusions
Simulation Environment

- Execution-driven simulator
- Scalable multiprocessor: 16 nodes
- Detailed superscalar processor model
- Processor: 4-issue, dynamic, 2K BTB
- 32 KB L1 2-way, 512 KB L2 4-way
- Speculative protocol [Zhang99]
Applications

- Applications dominated by non-analyzable loops (subscripted subscripts)
  - P3m (NCSA)
  - Tree (Univ. de Hawaii)
  - Apsi (Specfp2000)
  - Bdna (Perfect Club)
  - Track (Perfect Club)
  - Dsmc3d (HPF2)

- Non-analyzable loops and stores to instrument identified by the Polaris compiler

Non-analyzable loops account for an average of 51.4% of sequential time
Performance Results

- Sw only increases execution by 10% over Hw
- Sw reduces execution time by 36% over NoLog
Outline

- Speculative Parallelization
- Multiple Local Speculative Versions
- Software Logging
- Evaluation
- Conclusions
Conclusions

- Logs:
  - Support multiple versions
  - Minimize changes to cache

- Software logging:
  - No hardware support necessary
  - Low time overhead (10% over HW)

Software logging: good solution for spec parallelization
Software Logging under Speculative Parallelization

María Jesús Garzarán
(garzaran@posta.unizar.es)

M. Prvulovic, J. M. Llaberúa, V. Viñals, L. Rauchwerger, and J. Torrellas

http://www.cps.unizar.es/deps/DIIS/gaz
http://iacoma.cs.uiuc.edu
How to access Task_ID (TID)

- 2 special instructions: `lh_ts addr`, `sh_ts addr`
  where `addr` is virtual address of the data, not of the TID, since TIDs do not have virtual address

- `lh_ts`: bring data from TID page into cache
  `sh_ts`: update TID in cache

- Dependence-checking HW reads/updates the TID pages in memory automatically
Implementation of lh_ts Vaddress

2 possibilities:

- TLB has 2 physical addresses per entry
  
  | VaddressVar | PaddressVar | PaddressTID |

- TLB only has 1 physical address and there is a fixed offset between PaddressVar and PaddressTID
Hardware logging

- It has hardware cost:
  - FSM
  - Extra protocol messages
  - HW in caches to detect first writes...

- Need log physical address: complicates recovery
  - Should not have changed the mapping of Vir to Phys
  - Recovery needs to be done by privileged process
Instructions to insert in log

; r1 = upper limit of the sector
; r2 = address in memory to insert the log record
; offset(r3) = address of the variable to update

bgt    r1, r2, insertion
… allocate another sector

Logging instr

insertion:  

adddu  r4, r3, offset  ; address of the variable
sw      r4, 0(r2)      ; store in the log
lh_ts   r4, offset(r3) ; load the 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
adddu  r2, r2, log_record_size
sw      r5, offset(r3)
Reducing unnecessary instrumentation

- Not all the stores need to be instrumented
- Instrument:
  - first store of the non-speculative ones
  - all speculative stores
  - Run time filtering of the first speculative store
Filtering first speculative store

- Using Task-ID

```
lh_ts r6, offset (r3)  ; load task-ID
beq r6, r5, no_insert  ; first store?
addu r4, r3, offset    ; insert as usual
sw r4, 0(r2)
........
addu r2, r2, log_record_size
sh_ts r5, offset (r3)  ; store task-ID
sw r5, offset (r3)
```

Logging instr

no_insert:
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
Stores can cause squashes

- Stores can produce squashes of tasks that loaded a value prematurely
  - out-of-order RAW

Tasks 3 4 5 store 6 load

Cache
Memory
Network
3 Support for multiple versions

Tasks

Cache

Memory

Network
Logs help managing overflow area

- Logs hide away past versions of vars
- Overflow area and cache have the latest version
  - The processor will request the latest version
Problem: Address time stamp in software

- The time stamp is not mapped in virtual space
- How to make visible the time stamp to the sw?

Logging

\[
\begin{align*}
\text{lw} & \quad r3, \text{addr\_TS} \\
\text{sw} & \quad r5, \text{offset(r3)}
\end{align*}
\]

Undo Log

<table>
<thead>
<tr>
<th>Vaddr</th>
<th>Version</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem: Address time stamp in software

- OS copies
  - Data page in even page
  - Time stamp page in next odd page

Logging inst

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

Undo Log

<table>
<thead>
<tr>
<th>Vaddr</th>
<th>Version</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Log Sizes

<table>
<thead>
<tr>
<th>Appl</th>
<th>Log size/Task (KB)</th>
<th># Tasks in Undo Log per Processor (Recycle)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All</td>
<td>Filter</td>
</tr>
<tr>
<td>Apsi</td>
<td>184</td>
<td>40</td>
</tr>
<tr>
<td>P3m</td>
<td>56.7</td>
<td>18.2</td>
</tr>
<tr>
<td>Dsmc3d</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Track</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>
Logging under exposed loads

- A local version can be killed with an exposed load
- Hardware must detect it and send an interrupt

Tasks

- 3
- 4
- 5
- 6 load

Cache

Memory

Network
Loads find correct version

- On a exposed load
  - the speculation protocol finds the correct version
  - provides it to the consumer task

Tasks

3

4

5

6 load

Cache

Memory

Network