A Scalable Approach to Thread-Level Speculation

J. Gregory Steffan, Christopher B. Colohan,
Antonia Zhai, and Todd C. Mowry

Computer Science Department
Carnegie Mellon University
Multithreaded Machines Are Everywhere

How can we use them? Parallelism!

诮 A Scalable Approach to Thread-Level Speculation

Carnegie Mellon

Steffan
Automatic Parallelization

Proving independence of threads is hard:
- complex control flow
- complex data structures
- pointers, pointers, pointers
- run-time inputs

How can we make the compiler’s job feasible?

.Thread-Level Speculation (TLS)
Example

while (...){
    x = hash[index1];
    ...
    hash[index2] = y;
    ...
}

Time

Processor

= hash[3]
...
hash[10] =
...
= hash[19]
...
hash[21] =
...
= hash[33]
...
hash[30] =
...
= hash[10]
...
hash[25] =
Example of Thread-Level Speculation

Epoch 1
= hash[3]
... hash[10] = ...

Epoch 2
= hash[19]
... hash[21] = ...

Epoch 3
= hash[33]
... hash[30] = ...

Epoch 4
= hash[10]
... hash[25] = ...

A Scalable Approach to Thread-Level Speculation
Carnegie Mellon
Steffan
Example of Thread-Level Speculation

Epoch 1
= hash[3]
... hash[10] =
... commit?

Epoch 2
= hash[19]
... hash[21] =
... commit?

Epoch 3
= hash[33]
... hash[30] =
... commit?

Epoch 4
= hash[10]
... hash[25] =
... commit?

*Violation!*
Example of Thread-Level Speculation

Processor

Epoch 1
= hash[3]
... hash[10] = ... commit?

Epoch 2
= hash[19]
... hash[21] = ... commit?

Epoch 3
= hash[33]
... hash[30] = ... commit?

Epoch 4
= hash[10]
... hash[25] = ... commit?

Time

Violation!

Retry

Carnegie Mellon

A Scalable Approach to Thread-Level Speculation 7

Steffan
Goals of Our Approach

1) Handle arbitrary memory accesses
   - i.e. not just array references

2) Preserve performance of non-speculative workloads
   - keep hardware support minimal and simple

3) Apply to any scale of multithreaded architecture
   - CMPs, SMT processors, more traditional MPs

☞ effective, simple, and scalable TLS
Overview of Our Approach

System requirements:

1) Detect data dependence violations
   • extend invalidation-based cache coherence

2) Buffer speculative modifications
   • use the caches as speculative buffers

coherence already works at a variety of scales

hence our scheme is also scalable
Related Schemes

- Wisconsin (Multiscalar, Trace Processor)
- Stanford (Hydra)
- U.P. Catalunya (Speculative Multithreading)
- Intel/U. Portland (Dynamic Multithreading)
- Illinois at U.C. (I-ACOMA)

Our approach seamlessly scales both up and down
Outline

Details of our Approach

- life cycle of an epoch
- speculative coherence
- what happens at commit time
- forwarding data between epochs

- Performance

- Conclusions
Life Cycle of an Epoch

Time

Speculative Work

Init

Wait to be Homefree?

Spawning

Becomes Speculative

Commit?

Complete, Pass Homefree

Slow Commit:

Fast Commit:

Carnegie Mellon

A Scalable Approach to Thread-Level Speculation

Steffan
Life Cycle of an Epoch

Time

Spawned

Becomes Speculative

Speculative Coherence

Commit?

Complete, Pass Homefree

Mechanisms to Squash or Commit
MESI Coherence Example

Thread A:

Thread B:

Shared Memory (X=2)
MESI Coherence Example

Thread A:

Thread B:

Load X

Shared Memory (X=2)
A Scalable Approach to Thread-Level Speculation

Thread A:
Store X=3

Thread B:
Load X

 MESI Coherence Example

- **Processor**
  - **Cache**
    - State: Dirty
    - Tag: X
    - Data: 3
  - Read-Exclusive
  - Fill
  - Invalidation

- **Processor**
  - **Cache**
    - State: Invalid
    - Tag: -
    - Data: -

- **Shared Memory**
  - (X*)

- the state ‘dirty’ implies exclusiveness

Carnegie Mellon

Steffan
Speculative Coherence Example

Epoch 4:

Load X

Epoch 5:

Store X = 3

Epoch 6:

Load X

Highlights of our scheme:

- detection of a data dependence violation
- speculatively modified \textit{and} shared cache lines
Speculative Coherence Example

Epoch 5:

- Processor
- Cache
  - State: Invalid
  - Tag: -
  - Data: -

Epoch 6:

- Load X
- Processor
- Cache
  - State: Excl.
  - Tag: X
  - Data: 2

Shared Memory (X=2)

-track which lines are speculatively loaded
Speculative Coherence Example

Epoch 5: Store X=3

Processor

Cache
State: Invalid
Tag: -
Data: -

Spec. Read-Ex (epoch 5)

Shared Memory (X=2)

Epoch 6: Load X

Processor

Cache
State: Excl.
Tag: X
Data: 2

Spec. Inv (epoch 5)

(epoch 5 < epoch 6, and speculatively loaded)
Speculative Coherence Example

Epoch 5: Store X = 3

Epoch 6: Load X speculation failed!

Speculative Coherence Example

Track which lines are speculatively modified
Speculative Coherence Example

Highlights of our scheme:
- detection of a data dependence violation
- speculatively modified *and* shared cache lines
Speculative Coherence Example

**Epoch 4:**
- **Load X**
  - Processor
    - Cache
      - State: Shared
      - Tag: X
      - Data: 2
  - Spec. Loaded
  - Fill
  - Read

**Epoch 5:**
- **Store X=3**
  - Processor
    - Cache
      - State: Shared
      - Tag: X
      - Data: 3
  - Spec. Modified
  - notify shared

**Multiple versions of the same cache line**
Summary of New Speculative Line State

New cache line state:
- has it been *speculatively loaded*?
  - detect dependence violations
- has it been *speculatively modified*?
  - buffer speculative modifications
- is it in a *speculative shared* or *exclusive* state?
  - important performance optimizations

What if a speculative cache line is replaced?
- speculation fails for that epoch
Implementation of Speculative State

Speculatively Loaded

Speculatively Modified

modest amount of extra space
A Scalable Approach to Thread-Level Speculation

A Scalable Approach to Thread-Level Speculation

Life Cycle of an Epoch

Time

Speculative Coherence

邵wned

Becomes Speculative

Commit?

Complete, Pass Homefree

Mechanisms to Squash or Commit
### When Speculation Fails

A Scalable Approach to Thread-Level Speculation

**Carnegie Mellon**

**Steffan**

<table>
<thead>
<tr>
<th>SL</th>
<th>SM</th>
<th>State</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Sp Ex</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Sp Sh</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Sp Ex</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Sp Sh</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

**Diagram:**
- **Processor**
- **Cache**
- **Flash Reset**
When Speculation Fails

If Set then
Invalidate;
Flash Reset
### When Speculation Fails

**Quick Bit Operation**

![Cache Diagram]

<table>
<thead>
<tr>
<th>SL</th>
<th>SM</th>
<th>State</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Excl</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Shared</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Invalid</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>Invalid</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>
Life Cycle of an Epoch

Spawning

Becomes Speculative

Speculative Coherence

Commit?

Complete, Pass Homefree

Mechanisms to Squash or Commit

Time
When Speculation Succeeds

<table>
<thead>
<tr>
<th>SL</th>
<th>SM</th>
<th>State</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>Sp Ex</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Sp Sh</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>Sp Ex</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Sp Sh</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Flash Reset

* Flash
* Reset
When Speculation Succeeds

SM & Exclusive: Become Dirty
When Speculation Succeeds

- SM & Shared: Need Exclusive Access

Point: want to avoid searching entire cache
When Speculation Succeeds

Ownership required buffer (ORB)
When Speculation Succeeds

If SM, Become Dirty; Flash Reset

Ack (X) Upgrade-Request (X)
When Speculation Succeeds

flush the ORB, then quick bit operations
Forwarding Data Between Epochs

- predictable dependences cause frequent violations
- compiler inserts wait-signal synchronization

🚀 synchronize to avoid violations

Store X  Load X  

With Forwarding

Store X  Signal

Wait  Load X
Outline

- Details of our Approach

个人观点 Performance
- simulation infrastructure
- single-chip multiprocessor performance
- scaling beyond chip boundaries

- Conclusions
Simulation Infrastructure

Compiler system and tools based on SUIF
- help analyze dependences, insert synchronization
- produce MIPS binaries containing TLS primitives

Benchmarks (all run to completion)
- buk, compress95, ijpeg, equake

Simulator
- superscalar, similar to MIPS R10K
- models all bandwidth and contention

*detailed simulation!*
Performance on a 4-Processor CMP

Parallel Coverage:
- buk: 56.6%
- compress95: 47.3%
- equake: 39.3%
- ijpeg: 22.1%
Performance on a 4-Processor CMP

Parallel Coverage:
- **buk**: 56.6%
- **compress95**: 47.3%
- **equake**: 39.3%
- **jpeg**: 22.1%

Program speedups are limited by coverage.
Varying the Number of Processors

buk and equake are memory-bound

compress95 and jpeg are computation-intensive
Varying the Number of Processors

buk and equake scale well

passing the homefree token is not a bottleneck
Performance of the ORB (on a 4-CMP)

<table>
<thead>
<tr>
<th>Application</th>
<th>Average Flush Latency (cycles)</th>
<th>ORB Size (entries)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Average</td>
<td>Maximum</td>
</tr>
<tr>
<td>buk</td>
<td>13.95</td>
<td>2.38</td>
</tr>
<tr>
<td></td>
<td></td>
<td>9</td>
</tr>
<tr>
<td>compress95</td>
<td>0.04</td>
<td>0.01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8</td>
</tr>
<tr>
<td>equake</td>
<td>0.13</td>
<td>0.04</td>
</tr>
<tr>
<td></td>
<td></td>
<td>12</td>
</tr>
<tr>
<td>ijpeg</td>
<td>1.06</td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

✍️ a small ORB is sufficient
Tracking Dependences Per Cache Line

Problem:
- analogous to false sharing: false violations
- write-after-write dependences also cause violations
  - but not a true dependence!

Solution:
- track dependences at a word granularity
- have an SM and SL bit per word in each cache line

_quiz: is per-word state worth the extra overhead?_
Tracking Dependences Per Cache Line

Does it do any good?
- not for our 4 benchmarks
- adding this support showed no improvement

Why not?
- buk and equake have random access patterns
- compress95 is heavily synchronized
- jpeg is unrolled to avoid false sharing

 existing techniques for avoiding false sharing can address this problem
Scaling Beyond Chip Boundaries

simulate architectures with 1, 2 and 4 nodes
Scaling Beyond Chip Boundaries

multi-chip systems benefit from TLS
Scaling Beyond Chip Boundaries

Our scheme scales well.
Conclusions

The overheads of our scheme are low:
- mechanisms to squash or commit are not a bottleneck
- per-word speculative state is not always necessary

It offers compelling performance improvements:
- program speedups from 8% to 46% on a 4-processor CMP
- program speedups up to 75% on multi-chip architectures

It is scalable:
- coherence provides elegant data dependence tracking

⇒ seamless TLS on a wide range of architectures