Lecture 29(a)

Intro to Thread-Level Speculation
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;
    ...
}

Processor

Time

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

Time

Processor

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

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

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

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

Carnegie Mellon
Example of Thread-Level Speculation

<table>
<thead>
<tr>
<th>Processor</th>
<th>Epoch 1</th>
<th>Epoch 2</th>
<th>Epoch 3</th>
<th>Epoch 4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>hash[10] =</td>
<td></td>
<td>Violation!</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Time
Example of Thread-Level Speculation

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

Epoch 2
= hash[19]
... hash[21] = ...
commit? *Violation!*

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

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

Processor
Processor
Processor
Processor
Example of Thread-Level Speculation

<table>
<thead>
<tr>
<th>Processor</th>
<th>Processor</th>
<th>Processor</th>
<th>Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Epoch 1</td>
<td>Epoch 2</td>
<td>Epoch 3</td>
<td>Epoch 4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>hash[10] =</td>
<td>Violation!</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>commit?</td>
<td>commit?</td>
<td>commit?</td>
<td>commit?</td>
</tr>
</tbody>
</table>

Time

Carnegie Mellon
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
Life Cycle of an Epoch

Time

- Init
- Speculative Work
- Wait to be Homefree?
- Spawning
- Becomes Speculative
- Commit?
- Complete, Pass Homefree
- Slow Commit:
- Fast Commit:

Carnegie Mellon
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

็ด detailed simulation!
Performance on a 4-Processor CMP

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

Carnegie Mellon
Performance on a 4-Processor CMP

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

Program speedups are limited by coverage
Varying the Number of Processors

- **buk** and **equake** are memory-bound.
- **compress95** and **ijpeg** are computation-intensive.
Varying the Number of Processors

buk and equake scale well

passing the homefree token is not a bottleneck
Scaling Beyond Chip Boundaries

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

multi-chip systems benefit from TLS
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*]