Heuristics for Profile-driven Method-level Speculative Parallelization

John Whaley and Christos Kozyrakis
Stanford University

June 15, 2005
Speculative Multithreading

• Speculatively parallelize an application
  – Uses speculation to overcome ambiguous dependencies
  – Uses hardware support to recover from misspeculation
  – Promising technique for automatically extracting parallelism from programs

• Problem: Where to put the threads?
Method–Level Speculation

• Idea: Use method boundaries as speculative threads
  – Computation is naturally partitioned into methods
  – Execution often independent
  – Well–defined interface

• Extract parallelism from irregular, non–numerical applications
Method-Level Speculation Example

```c
main()
{
    work_A;

    foo();

    work_C;  // reads *q
}

foo()
{
    work_B;  // writes *p
}
```
Method–Level Speculation Example

```c
main()
{
    work_A;
    foo() {
        work_B; // writes *p
    }
    work_C; // reads *q
}
```
main()
{
    work_A;
    foo() {
        work_B; // writes *p
    }
    work_C; // reads *q
}
Method-Level Speculation Example

main()
{
    work_A;
    foo() {
        work_B; // writes *p
    }
    work_C; // reads *q
}

TLS execution – no violation

work_A
fork

foo()
work_B

overhead

work_C

p!=q
No violation

TLS execution – no violation
main()
{
    work_A;
    foo() {
        work_B; // writes *p
    }
    work_C; // reads *q
}

TLS execution – violation

Method-Level Speculation Example

June 15, 2005 Heuristics for Profile-driven Method-level Speculative Parallelization
Method-Level Speculation Example

Sequential

work_A

foo()

work_B

work_C

TLS – no violation

work_A

fork

foo()

work_B

overhead

work_C

p!=q
No violation

TLS – violation

work_A

fork

foo()

work_B

overhead

work_C

p=q
Violation!

work_C (aborted)
Nested Speculation

Sequences of method calls can cause nested speculation.

June 15, 2005

Heuristics for Profile-driven Method-level Speculative Parallelization
This Talk: Choosing Speculation Points

• Which methods to speculate?
  – Low chance of violation
  – Not too short, not too long
  – Not too many stores

• Idea: Use profile data to choose good speculation points
  – Used for profile-driven and dynamic compiler
  – Should be low-cost but accurate

• We evaluated 7 different heuristics
  – ~80% effective compared to perfect oracle
Difficulties in Method-Level Speculation

- Method invocations can have varying execution times
  - Too short: Doesn’t overcome speculation overhead
  - Too long: More likely to violate or overflow, prevents other threads from retiring
- Return values
  - Mispredicted return value causes violation
Classes of Heuristics

• Simple Heuristics
  – Use only simple information, such as method runtime

• Single-Pass Heuristics
  – More advanced information, such as sequence of store addresses
  – Single pass through profile data

• Multi-Pass Heuristics
  – Multiple passes through profile data
Classes of Heuristics

- **Simple Heuristics**
  - Use only simple information, such as method runtime

- **Single-Pass Heuristics**
  - More advanced information, such as sequence of store addresses
  - Single pass through profile data

- **Multi-Pass Heuristics**
  - Multiple passes through profile data
Runtime Heuristic (SI–RT)

• Speculate on all methods with:
  – MIN < runtime < MAX

• Idea: Should be long enough to amortize overhead, but not long enough to violate

• Data required:
  – Average runtime of each method
Store Heuristic (SI–SC)

- Speculate on all methods with:
  - dynamic # of stores < MAX
- Idea: Stores cause violations, so speculate on methods with few stores
- Data required:
  - Average dynamic store count of each method
Classes of Heuristics

• Simple Heuristics
  – Use only simple information, such as method runtime

• Single-Pass Heuristics
  – More advanced information, such as sequence of store addresses
  – Single pass through profile data

• Multi-Pass Heuristics
  – Multiple passes through profile data
Stalled Threads

Speculative threads may stall while waiting to become main thread.

June 15, 2005
Heuristics for Profile-driven Method-level Speculative Parallelization
Fork at intermediate points

foo()
{
    bar() {
        work_A;
    }
    work_B;
}

Fork at an intermediate point within a method to avoid violations and stalling
Best Speedup Heuristic (SP–SU)

- Speculate on methods with:
  - predicted speedup > THRES
- Calculate predicted speedup by:
  - expected sequential run time
  - expected parallel run time
- Scan store stream backwards to find fork point
  - Choose fork point to avoid violations and stalling
Most Cycles Saved Heuristic (SP–CS)

• Speculate on methods with:
  – predicted cycle savings > THRES
• Calculate predicted cycle savings by:
  sequential cycle count – parallel cycle count
• Place fork point such that:
  – predicted probability of violation < RATIO
• Uses same information as SP–SU
Classes of Heuristics

• Simple Heuristics
  – Use only simple information, such as method runtime

• Single-Pass Heuristics
  – More advanced information, such as sequence of store addresses
  – Single pass through profile data

• Multi-Pass Heuristics
  – Multiple passes through profile data
Nested Speculation

main()
{
    foo() {
        work_A;
        bar() {
            work_B;
        }
        work_C;
    }
    work_D;
}

Effectiveness of speculation choice depends on choices for caller methods!
Best Speedup Heuristic with Parent Info (MP–SU)

• Iterative algorithm:
  – Choose speculation with best speedup
  – Readjust all callee methods to account for speculation in caller
  – Repeat until best speedup < THRES

• Max # of iterations: depth of call graph
Most Cycles Saved Heuristic with Parent Info (MP-CS)

- Iterative algorithm:
  1. Choose speculation with most cycles saved and predicted violations < RATIO
  2. Readjust all callee methods to account for speculation in caller
  3. Repeat until most cycles saved < THRES

- Multi-pass version of SP-CS
Most Cycles Saved Heuristic with No Nesting (MP-CSNN)

- **Iterative algorithm:**
  - Choose speculation with most cycles saved and predicted violations < RATIO.
  - *Eliminate* all callee methods from consideration.
  - Repeat until most cycles saved < THRES.

- **Disallows nested speculation to avoid double-counting the benefits**
- **Faster to compute than MP-CS**
Experimental Results
Trace–Driven Simulation

- How to find the optimal parameters (THRES, RATIO, etc.)?
- Parameter sweeps
  - For each benchmark
    - For each heuristic
      - Multiple parameters for each heuristic
- For cycle-accurate simulation: >100 CPU years?!
- Alternative: trace–driven simulation
Trace-Driven Simulation

1. Collect trace on Pentium III (3-way out-of-order CPU, 32K L1, 256K L2)
   - Record all memory accesses, enter/exit method events, etc.
2. Recalibrate to remove instrumentation overhead
3. Simulate trace on 4-way CMP hardware
   - Model shared cache, speculation overheads, dependencies, squashing, etc.
Spot check with cycle-accurate simulator: Accurate within ~3%
Simulated Architecture

- Four 3-way out-of-order CPUs
  - 32K L1, 256K shared L2
- Single speculative buffer per CPU
- Forking, retiring, squashing overhead: 70 cycles each
- Speculative threads can be preempted
  - Low priority speculations can be squashed by higher priority ones
The Oracle

- A “Perfect” Oracle
  - Preanalyzes entire trace
  - Makes a separate decision on every method invocation
  - Chooses fork points to never violate
  - Zero overhead for forking or retiring threads

- Upper-bound on performance of any heuristic
Benchmarks

• SpecJVM
  – compress: Lempel–Ziv compression
  – jack: Java parser generator
  – javac: Java compiler from the JDK 1.0.2
  – jess: Java expert shell system
  – mpeg: Mpeg layer 3 audio decompression
  – raytrace: Raytracer that works on a dinosaur scene

• SPLASH–2
  – barnes: Hierarchical N–body solver
  – water: Simulation of water molecules
Heuristic Parameter Tuning

Graphs showing the relationship between speedup and runtime, with varying parameter values. The left graph illustrates speedup, while the right graph shows the number of violations. Both graphs feature a 3D representation with axes labeled for MAX and MIN values, ranging from $10^3$ to $10^{12}$. The graphs demonstrate the impact of different parameter settings on performance metrics.
Heuristic Parameter Tuning

Store (SI-SC)

- Speedup vs. Threshold
- Bars for Void only, Constant, and Perfect

- Speedup values: 1.00, 1.10, 1.20, 1.30, 1.40, 1.50
- Threshold values: 1e1, 1e2, 1e3, 1e4, 1e5
Heuristic Parameter Tuning

Store (SI-SC)

Number of violations

Threshold

0 1e1 1e2 1e3 1e4 1e5

1e1 1e2 1e3 1e4 1e5

Threshold

Store (SI-SC)

Number of violations

0 1e1 1e2 1e3 1e4 1e5

Threshold

Violet only
Constant
Perfect
Heuristics for Profile-driven Method-level Speculative Parallelization

Heuristic Parameter Tuning

Best Speedup (SP-SU)

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Void only</th>
<th>Constant</th>
<th>Perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.001</td>
<td>1.50</td>
<td>1.40</td>
<td>1.50</td>
</tr>
<tr>
<td>1.01</td>
<td>1.50</td>
<td>1.40</td>
<td>1.50</td>
</tr>
<tr>
<td>1.1</td>
<td>1.50</td>
<td>1.40</td>
<td>1.50</td>
</tr>
<tr>
<td>1.2</td>
<td>1.50</td>
<td>1.40</td>
<td>1.50</td>
</tr>
<tr>
<td>1.4</td>
<td>1.50</td>
<td>1.40</td>
<td>1.50</td>
</tr>
<tr>
<td>1.6</td>
<td>1.50</td>
<td>1.40</td>
<td>1.50</td>
</tr>
</tbody>
</table>
Heuristic Parameter Tuning

Best Speedup (SP-SU)

Number of Violations

Threshold

0 50 100 150 200 250 300 350

1.001 1.01 1.1 1.2 1.4 1.6

Void only
Constant
Perfect
Heuristic Parameter Tuning

Most Cycles Saved (SP-CS)

Speedup

0.1 0.3 0.5 0.7 0.9 1e2 1e5 1e7

RATIO

THRES

0 50 100 150 200 250 300 350

Number of violations

0.1 0.3 0.5 0.7 0.9 1e2 1e5 1e7

RATIO

THRES

June 15, 2005

Heuristics for Profile-driven Method-level Speculative Parallelization
Heuristic Parameter Tuning

Best Speedup with Parent Info (MP-SU)

<table>
<thead>
<tr>
<th>Threshold</th>
<th>Void only</th>
<th>Constant</th>
<th>Perfect</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.001</td>
<td>1.1</td>
<td>1.2</td>
<td>1.4</td>
</tr>
<tr>
<td>1.01</td>
<td>1.15</td>
<td>1.25</td>
<td>1.4</td>
</tr>
<tr>
<td>1.1</td>
<td>1.15</td>
<td>1.25</td>
<td>1.4</td>
</tr>
<tr>
<td>1.2</td>
<td>1.2</td>
<td>1.25</td>
<td>1.4</td>
</tr>
<tr>
<td>1.4</td>
<td>1.3</td>
<td>1.35</td>
<td>1.5</td>
</tr>
<tr>
<td>1.6</td>
<td>1.4</td>
<td>1.45</td>
<td>1.6</td>
</tr>
</tbody>
</table>
Heuristic Parameter Tuning

Best Speedup with Parent Info (MP-SU)

Number of violations

Threshold

1.001 1.01 1.1 1.2 1.4 1.6

0 100 200 300 400 500

Void only
Constant
Perfect
Heuristic Parameter Tuning

Most Cycles Saved with Parent Info (MP-CS)

Most Cycles Saved with Parent Info (MP-CS)
Heuristics for Profile-driven Method-level Speculative Parallelization
Tuning Summary

• Runtime (SI–RT):
  – MIN = 10^3 cycles, MAX = 10^7 cycles
• Store (SI–SC):
  – MAX = 10^5 stores
• Best speedup (SP–SU, MP–SU):
  – Single pass: MIN = 1.2x speedup
  – Multi pass: MIN = 1.4x speedup
• Most cycles saved (SP–CS, MP–CS, MP–CSNN):
  – THRES = 10^5 cycles saved, RATIO = 70% violation
• Return value prediction:
  – Constant is within 15% of perfect value prediction
Overall Speedups

- Overall Speedups

- barnes
- compress
- jack
- javac
- jess
- mpeg
- raytrace
- water
- Average

- SI-RT
- SI-SC
- SP-SU
- SP-CS
- MP-SU
- MP-CS
- MP-CSNN
- Oracle
Breakdown of Speculative Threads

Normalized number of threads

Successful  Preempted  Killed

barnes  compress  jack  javac  jess  mpeg  raytrace  water

June 15, 2005  Heuristics for Profile-driven Method-level Speculative Parallelization
Breakdown of Execution Time

Normalized Execution Time

Useful □ Idle □ Wasted

June 15, 2005

Heuristics for Profile-driven Method-level Speculative Parallelization
### Speculative Store Buffer Size

<table>
<thead>
<tr>
<th></th>
<th>barnes</th>
<th>comp</th>
<th>jack</th>
<th>javac</th>
<th>jess</th>
<th>mpeg</th>
<th>rtrace</th>
<th>water</th>
</tr>
</thead>
<tbody>
<tr>
<td>SI–RT</td>
<td>0.31</td>
<td>0.18</td>
<td>0.39</td>
<td>2.05</td>
<td>0.26</td>
<td>0.76</td>
<td>1.64</td>
<td>0.20</td>
</tr>
<tr>
<td>SI–SC</td>
<td>12.02</td>
<td>6.47</td>
<td>0.19</td>
<td>3.51</td>
<td>0.15</td>
<td>13.02</td>
<td>1.64</td>
<td>1.45</td>
</tr>
<tr>
<td>SP–SU</td>
<td>8.11</td>
<td>6.48</td>
<td>0.39</td>
<td>1.08</td>
<td>0.30</td>
<td>13.02</td>
<td>1.64</td>
<td>0.55</td>
</tr>
<tr>
<td>SP–CS</td>
<td>0.31</td>
<td>6.48</td>
<td>0.39</td>
<td>2.57</td>
<td>0.30</td>
<td>15.29</td>
<td>1.64</td>
<td>0.22</td>
</tr>
<tr>
<td>MP–SU</td>
<td>12.01</td>
<td>6.48</td>
<td>0.39</td>
<td>0.30</td>
<td>0.30</td>
<td>1.27</td>
<td>1.27</td>
<td>1.38</td>
</tr>
<tr>
<td>MP–CS</td>
<td>12.02</td>
<td>6.48</td>
<td>0.39</td>
<td>0.30</td>
<td>0.30</td>
<td>1.64</td>
<td>1.27</td>
<td>1.38</td>
</tr>
<tr>
<td>MP–CSNN</td>
<td>12.02</td>
<td>6.48</td>
<td>0.39</td>
<td>2.57</td>
<td>0.30</td>
<td>13.02</td>
<td>1.27</td>
<td>1.38</td>
</tr>
</tbody>
</table>

**Maximum speculative store buffer size: 16KB**
Related Work

• Loop–level parallelism
• Method–level parallelism
  – Warg and Stenstrom
    • ICPAC’01: Limit study
    • IPDPS’03: Heuristic based on runtime
    • CF’05: Misspeculation prediction

• Compilers
  – Multiscalar: Vijaykumar and Sohi, JPDC’99
  – SpMT: Bhowmik & Chen, SPAA’02
Conclusions

• Evaluated 7 heuristics for method-level speculation

• Take-home points:
  – Method-level speculation has complex interactions, very hard to predict
  – Single-pass heuristics do a good job: 80% of a perfect oracle
  – Most important issue is the balance between over- and under-speculating