Speculative Multithreaded Processors

Guri Sohi and Amir Roth

Computer Sciences Department
University of Wisconsin-Madison
Outline

• Trends and their implications
• Workloads for future processors
• Program parallelization and speculative threads
  ◦ speculative control-driven threads
  ◦ speculative data-driven threads
• Sample applications and research issues
• Summary
Driving Factors

• Match upcoming technology trends
• Match upcoming software trends
• Match upcoming technology constraints
• Match upcoming design constraints
• Learn, and exploit, new program behaviors
Hardware/Design Trends

- Increasing wire delays
- Increasing memory latencies
- Deeper pipelines
- Design complexity
- Verification complexity
- Power issues
Implications of Trends

• Distributed microarchitectures
• Clustered superscalar, with multithreading
• Chip multiprocessor

Question: what to run on underlying microarchitecture?
Work for Distributed/Multithreaded Processor

• Independent programs
  ◦ increase overall processing throughput
  ◦ works well in server environment
• Independent threads of multithreaded application
  ◦ increase overall throughput
  ◦ compatible with software trends?
• Related threads
  ◦ e.g., for reliability
• But what about speeding up single program execution?
  ◦ single program speed will continue to be important
  ◦ how to “parallelize” or “multithread” single program?
Program Parallelization

• What does it mean to parallelize?
  o how to divide program into multiple portions

• What constrains parallelization?
  o dependences (especially ambiguous)

• How to overcome constraints?
  o use speculation
Program Parallelization -- Theme 1

- **Traditional view:** control-driven threads
  - divide work into multiple groups of instructions
    - conservative assumptions about dependences constrain parallelization
  - each group is specified using traditional control-driven (von Neumann) semantics

- **A newer view:** multiscalar
  - use dependence speculation to overcome constraints
  - commercial example: Sun MAJC
Multiscalar: Speculative Control-Driven Threads

PROGRAM

PROC UNIT 1

predict

PROC UNIT 2

predict

PROC UNIT 3

A

B

C

A

B

C

Speculative Multithreaded Processors
Program Parallelization -- Theme II

• Another traditional view: dataflow
  ◦ divide work into (dependent) computations
  ◦ each computation is represented in a data-driven manner

• A newer view: speculative data-driven “threads”
  ◦ use speculation to facilitate thread creation
Motivation for Data-driven Threads

- program execution: processing of low-latency instructions, with pauses for high-latency events
- parallelizing low-latency instructions isn’t crucial
- overlapping high-latency events is what matters!
- “threads” should create high-latency events early
Speculative Data-Driven Threads

- Use dependence relationships to isolate thread(s) of code from main program thread
  - use speculation to facilitate creation
- Execute threads (speculatively) in parallel with “main program”
  - “assist” main thread via side-effects
  - don’t impact architectural correctness
### Application: Cache Misses and Branch Mispredicts

<table>
<thead>
<tr>
<th>Spec2000 Benchmark</th>
<th>Memory</th>
<th>Control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td># inst</td>
<td>% dyn. memops</td>
</tr>
<tr>
<td>bzip2</td>
<td>24</td>
<td>3</td>
</tr>
<tr>
<td>crafty</td>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>eon</td>
<td>Insufficient misses</td>
<td></td>
</tr>
<tr>
<td>gap</td>
<td>66</td>
<td>1</td>
</tr>
<tr>
<td>gcc</td>
<td>122</td>
<td>4</td>
</tr>
<tr>
<td>gzip</td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>mcf</td>
<td>42</td>
<td>35</td>
</tr>
<tr>
<td>parser</td>
<td>70</td>
<td>4</td>
</tr>
<tr>
<td>perl</td>
<td>74</td>
<td>1</td>
</tr>
<tr>
<td>twolf</td>
<td>116</td>
<td>7</td>
</tr>
<tr>
<td>vortex</td>
<td>71</td>
<td>1</td>
</tr>
<tr>
<td>vpr (route)</td>
<td>55</td>
<td>13</td>
</tr>
</tbody>
</table>
Perfecting a small set of instructions provides significant performance much of that of a perfect branch predictor and data cache.
Using Speculative Data-driven Threads

Pre-execution
FORK

Avoid Misprediction
Sample Performance Results

VPR (route)

- 200M instruction sample (starting at 14.1B on 20B run)
- 100M instruction warm-up for caches/predictors

32% SPEEDUP: 16% FROM PRE-FETCHING, 16% FROM BRANCHES

<table>
<thead>
<tr>
<th>Cache Misses (primary L1)</th>
<th>Branch Mispredictions</th>
</tr>
</thead>
<tbody>
<tr>
<td>number</td>
<td>rate</td>
</tr>
<tr>
<td>base</td>
<td>2,850,000</td>
</tr>
<tr>
<td>w/slices</td>
<td>1,340,000</td>
</tr>
</tbody>
</table>

- The full latency of the misses and mispredictions is not always hidden
Sample Applications

- Cache prefetching/management
- Computing branch outcomes
- TLB prefetching/management
- I/O prefetching
- Multiprocessor communication management
Some Research Issues

• How to divide control-driven program into data-driven threads?
• When to divide program?
• How to represent data-driven threads?
• Managing mixed thread workloads
Summary

• Hardware and design trends will lead to distributed/multithreaded processors
• Many options for running different thread types on underlying microarchitecture
• Overcome constraints to traditional “parallelization” techniques with speculation
  ◦ speculative control-driven threads
  ◦ speculative data-driven threads
• Most of the research still needs to be done