Instruction Fetch
using
Multiple Sequencers

Paramjit Oberoi
Gurindar Sohi

University of Wisconsin - Madison

August 19, 2002

The 2002 International Conference on Parallel Processing
Vancouver, Canada
Problem: Instruction Stream Discontinuities

- Discontinuities in the instruction stream
  - Taken Branches
  - Cache Line Boundaries
- Discontinuities limit fetch throughput
- Solutions
  - Remove discontinuities
  - Handle discontinuities
Known Solutions

- **Remove Discontinuities**
  - Cache Layout
    Example: Trace Cache
  - Code Layout
    Example: Software TC, Dynamo

- **Handle Discontinuities**
  - Collapsing Buffer
Wide Fetch is Inefficient

- Example: Trace Cache

<table>
<thead>
<tr>
<th>Trace 1</th>
<th>Trace 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 cycles</td>
<td>2 cycles</td>
</tr>
<tr>
<td>20 cycles (12 instructions)</td>
<td></td>
</tr>
</tbody>
</table>

- Consecutive traces **execute concurrently** (almost)
  - Instructions are executed in dataflow order
  - But they are **fetched serially**

- A few critical instructions are needed first
  - Wide fetch also fetches the intervening instructions
Out-of-Order Fetch

- Instructions are not needed in sequential order
- Can we fetch instructions in a more efficient order?

Analogy:

<table>
<thead>
<tr>
<th>Out-of-Order Execution</th>
<th>Out-of-Order Fetch</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uses execution resources more efficiently</td>
<td>Use fetch bandwidth more efficiently</td>
</tr>
<tr>
<td>Tolerates long latencies</td>
<td>Tolerate fetch delays</td>
</tr>
<tr>
<td>Higher execution throughput</td>
<td>Achieve high fetch throughput</td>
</tr>
</tbody>
</table>
Multiple Instruction Sequencers

- **Wide Fetch**
  - Fetch a large contiguous block of instructions

- **Multiple Fetch**
  - Fetch discontiguous blocks from multiple locations
Benefits of Multiple Sequencers

• Fetch throughput not limited by width of single sequencer

• Latency Tolerance
  • One cache miss does not block all sequencers
  • Overlap cache-miss latencies

• Flexibility
  • Not limited to sequential fetch
  • Fine-grain allocation of fetch bandwidth to threads
  • May ease implementation of many techniques
    • Dual-path execution
    • Speculative threads
Design Details
Single Sequencer

Instruction Cache → Sequencer → Instruction Fetch Queue → Decode Rename Execute Commit

Branch Predictor
Multiple Sequencers

- Trace predictor predicts future traces
- Each sequencer is assigned a trace to fetch
- Multiple sequencers operate in parallel
  - I-Cache has multiple banks
Each sequencer can write to any trace buffer

Instructions from oldest trace are placed in the IFQ

Steady State: Just-in-time trace construction
  - Storage efficiency of Instruction Cache
  - Performance of Trace Cache
Trace Reuse

- Reuse traces instead of constructing them again
  - Like a small trace cache

- 20%-70% traces can be reused with only 16 trace buffers
  - Reduced I-cache traffic
  - Potential performance & power benefits

Instruction Cache • Sequencers • Trace Buffers • Instruction Fetch Queue • Decode Rename Execute Commit
Trace Selection & Prediction

- Previously known techniques
  - Predictor: Breach [PhD Thesis], Jacobson et.al. [MICRO 1997]
  - 95% accuracy on average

- Trace selection heuristics
  - Maximum size is 16 instructions
  - End traces at Call/Return/SysCall
  - End traces at Unconditional Branchs if TraceSize > 8
    - Limit number of potential starting points - reduce working set
    - TraceSize > 8 increases average trace length
## Trace Characteristics

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Dynamic Instructions</th>
<th>Traces</th>
<th>Average Trace Size</th>
<th>Dynamic Traces</th>
<th>Traces Contributing 95% instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Integer</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>bzip2</td>
<td>8822 M</td>
<td>1819</td>
<td>12.79</td>
<td>690 M</td>
<td>109 (6%)</td>
</tr>
<tr>
<td>crafty</td>
<td>4265 M</td>
<td>7541</td>
<td>12.02</td>
<td>355 M</td>
<td>909 (12%)</td>
</tr>
<tr>
<td>gap</td>
<td>1246 M</td>
<td>9074</td>
<td>10.70</td>
<td>117 M</td>
<td>972 (11%)</td>
</tr>
<tr>
<td>gcc</td>
<td>2016 M</td>
<td>38180</td>
<td>11.26</td>
<td>179 M</td>
<td>7165 (19%)</td>
</tr>
<tr>
<td>gzip</td>
<td>3367 M</td>
<td>1942</td>
<td>12.06</td>
<td>279 M</td>
<td>58 (3%)</td>
</tr>
<tr>
<td>mcf</td>
<td>260 M</td>
<td>1424</td>
<td>9.84</td>
<td>26 M</td>
<td>132 (9%)</td>
</tr>
<tr>
<td>parser</td>
<td>4203 M</td>
<td>6496</td>
<td>10.35</td>
<td>406 M</td>
<td>692 (11%)</td>
</tr>
<tr>
<td><strong>Floating Point</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ammp</td>
<td>5491 M</td>
<td>2932</td>
<td>13.11</td>
<td>419 M</td>
<td>332 (11%)</td>
</tr>
<tr>
<td>equake</td>
<td>1443 M</td>
<td>2182</td>
<td>11.10</td>
<td>130 M</td>
<td>356 (16%)</td>
</tr>
<tr>
<td>lucas</td>
<td>3689 M</td>
<td>1090</td>
<td>15.68</td>
<td>235 M</td>
<td>130 (7%)</td>
</tr>
<tr>
<td>mesa</td>
<td>2845 M</td>
<td>2543</td>
<td>11.30</td>
<td>252 M</td>
<td>110 (4%)</td>
</tr>
</tbody>
</table>

- Average Trace Size ~ 10-12 instructions
- Less than 1000 traces contribute to 95% dynamic instructions
  - except gcc
Performance Evaluation
Configurations

- 16-wide processor, 256 entry instruction window
  - 64K L1, 256K L2

- W16 — conventional 16-wide fetch
  - stop at taken branches and cache-line boundaries

- MS-2x8w — Multiple Sequencers
  - Two 8-wide sequencers
  - 16 trace buffers of 16 instructions each

- TC — Trace Cache
  - 2-way set associative, 16 instructions per trace
  - 32K (512 lines) + 32K I-cache
  - I-cache + TC performs better than just TC
• 5% - 15% speedup over W16 on most benchmarks

• Performance similar to Trace Cache

• More efficient utilization of cache space (gcc)
- Number of instructions fetched from the I-Cache
  - Independent of cache-line size
- 20% increase over W16
Effect of Reuse on I-Cache Traffic

Committed

I-Cache Traffic without Reuse

I-Cache Traffic with Reuse

Out-of-Order Fetch using Multiple Sequencers
Effect of Reuse on Performance

Performance without Reuse

- bzip2
- crafty
- gap
- gcc
- gzip
- mcf
- parser
- ammp
- equake
- lucas
- mesa

Avg

Normalized Execution Time

0.6 0.7 0.8 0.9 1.0

Performance with Reuse

- bzip2
- crafty
- gap
- gcc
- gzip
- mcf
- parser
- ammp
- equake
- lucas
- mesa

Avg

Normalized Execution Time

0.6 0.7 0.8 0.9 1.0
Latency Tolerance

- Increase cache misses by reducing cache size
  - W16 and TC - performance deteriorates rapidly
  - MS performs robustly across a range of sizes

- MS is 20% faster on average for small cache sizes
Conclusion

• High bandwidth fetch
  • Existing solutions attempt wide fetch
  • Multiple fetch is also an alternative

• Multiple Sequencers
  • Not limited by instruction stream discontinuities
  • Performance of Trace Cache, without wasted storage
  • More latency tolerant than existing techniques

• Better fit for the future
  • Flexible allocation of fetch resources to threads
  • Robust across a wide range of cache miss rates
    (Smaller caches, Large working sets)
  • May make other optimizations easier
    (Dual-path execution, Speculative threads)
Backup: I-Cache Traffic

Out-of-Order Fetch using Multiple Sequencers

Slide 23 / 26
Out-of-Order Fetch using Multiple Sequencers
Backup: Latency Tolerance

- Increase cache misses by reducing cache size
  - W16 and TC - performance deteriorates rapidly
  - MS performs robustly across a range of sizes
- MS is 20% faster on average for small cache sizes
## Backup: Simulation Parameters

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Width</strong></td>
<td>Fetch, decode and commit at most 16 instructions per cycle</td>
</tr>
<tr>
<td><strong>Functional Units</strong></td>
<td>16 integer ALUs, 4 integer multipliers, 4 floating point ALUs, 1 floating point multiplier, 4 load/store units</td>
</tr>
</tbody>
</table>
| **In-flight Instructions** | 256 entry instruction window  
128 entry load/store queue |
| **L1 Caches**  | 64K, 2-way set-associative, 1 cycle access time, 64b blocks                |
| (Insn & Data)  |                                                                             |
| **L2 Cache**   | 256K, 4-way set-associative, 10 cycle access time, 128 byte blocks         |
| (Unified)      |                                                                             |
| **Memory**     | 100 cycle access time                                                       |
| **Trace Predictor** | 64K entry primary table  
16K entry secondary table  
D=9, O=4, L=7, C=9 |