Performance Prediction Models

Shruti Padmanabha, Andrew Lukefahr, Reetuparna Das, Scott Mahlke
{shrupad,lukefahr,reetudas,mahlke}@umich.edu

Gem5 workshop
Micro 2012
December 2, 2012
Composite Cores

• Pushing heterogeneity into a core

• A tightly coupled o3 backend (big) and an inorder one (little)
  – Big – 3 wide OoO with large ROB, LSQ
  – Little – 2 wide inorder, modeled as a OoO core with simplified pipeline, small ROB, no LSQ

• Switch at fine granularity or quantum (controller)

For more details, please attend the paper presentation on Tuesday
Operation of Composite Cores

Controller → "Run on big"

Little backend

Big backend

CPI = ???

Compare

Active!

CPI = 2.33
Objectives

• Run a quantum on one backend microarchitecture and project its performance on a different one dynamically

• **Challenge**: Only one is active at any given time

• **Solution**: Use a linear model to calculate the inactive core’s performance using the slice’s computational traits

\[ y = a_0 + \sum a_i x_i \]
# Performance defining factors

<table>
<thead>
<tr>
<th>Computational trait</th>
<th>Big</th>
<th>Little</th>
<th>Rel Performance Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>Independent chain of instructions (high ILP)</td>
<td>Exploits larger superscalar width</td>
<td>Lower throughput</td>
<td>High</td>
</tr>
<tr>
<td>Dependent chain of instructions (low ILP)</td>
<td>Issues in order</td>
<td>Issues in order</td>
<td>Low</td>
</tr>
<tr>
<td>Branch mispredictions</td>
<td>Large drain time</td>
<td>Smaller drain time</td>
<td>High</td>
</tr>
<tr>
<td>Independent chain of L2 misses (high MLP)</td>
<td>Can have multiple outstanding loads</td>
<td>Stalls</td>
<td>High</td>
</tr>
<tr>
<td>Dependent chain on L2 misses (low MLP)</td>
<td>Stalls</td>
<td>Stalls</td>
<td>Low</td>
</tr>
<tr>
<td>Icache misses</td>
<td>Stalls</td>
<td>Stalls</td>
<td>Low</td>
</tr>
</tbody>
</table>
Performance defining Counters

- Per program slice, dynamically track
  - Active CPI
  - # of Branch misses
  - # of L1 misses
  - # of L2 misses
  - # of Icache misses
  - ILP
  - MLP

- Append dynamic instruction class with fields that identify above parameters
Performance counters example

• Branch misses:
  – Set flag on branch mispredict discovery in iew_impl.hh

• Cache level:
  – Append the Packet class with a field to track the level of cache that satisfied request
  – Set field on packet return, in lsq_unit_impl.hh

```
lsq_unit_impl.hh
writeback(inst, pkt):
  inst->cachelevel = pkt->cachelevel
```

```
cache_impl.hh
access(pkt):
  L1?
  No, forward pkt
  L2?
  Yes
  pkt->cachelevel = L2
```
Measuring ILP & MLP – Big backend

While on big,

- Measure of ILP in a quantum:
  - Track # of instructions that are stalled due to dependencies in inst_queue_impl.hh

- Measure of MLP in a quantum:
  - Track # of MSHR entries in use at each L1 cache miss
Measuring ILP & MLP – Little backend

While on little,

• More complicated, since the smaller core doesn't have the ability to exploit these characteristics

• Track data dependence chains for the window of instructions that big cares about (it’s ROB length)
Little’s Dependence Tracker Table*

- Bit matrix – (#ROB entries in big) * (#registers)
- Implemented using multidimensional vectors in commit_impl.hh
- Passes information about the serialization inherent in code section to the instruction class

*L. Chen, S. Dropsho, and D. Albonesi, “Dynamic data dependence tracking and its application to branch prediction”*
# Dependence Tracker Table

<table>
<thead>
<tr>
<th>Inst #</th>
<th>r1</th>
<th>r2</th>
<th>r3</th>
<th>r4</th>
<th>r5</th>
<th>r6</th>
<th>r7</th>
<th>r8</th>
<th>Instruction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inst 1</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>r2 &lt;- r3 + r5</td>
</tr>
<tr>
<td>Inst 2</td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r4 &lt;- r1 – r2</td>
</tr>
<tr>
<td>Inst 3</td>
<td>X</td>
<td>Miss</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r1 &lt;- Load(r7)</td>
</tr>
<tr>
<td>Inst 4</td>
<td>X</td>
<td>Miss</td>
<td>Miss</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Miss</td>
<td>r3 &lt;- Load(r2)</td>
</tr>
<tr>
<td>Inst 5</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>r2 &lt;- r3 or r6</td>
</tr>
<tr>
<td>Inst 6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>Miss</td>
<td>r8 &lt;- Load(r2)</td>
</tr>
</tbody>
</table>

8 register system, with 7 big ROB entries

2 independent, overlappable and 1 dependent L2 miss seen in this window.
Max available MLP = 2
Prediction Accuracy

big->little

Regression

95% of the quantums predicted with <10% error

little->big

Regression

82% of the quantums predicted with <10% error
Conclusion

• Implemented a performance prediction model by measuring characteristics like MLP and ILP in a code section with prediction accuracy of 88.5% on average

• Gem5 modifications:
  – Created a new controller class to carry out prediction calculations dynamically. This was invoked on every instruction commit in commit_impl.hh
  – New fields added to the base_dyn_inst class
  – Tracker table implemented in commit_impl.hh in commitInsts()
  – Counters were also added in inst_queue_impl, iew_impl, cache_impl, lsq_unit_impl

• Total lines of code added: ~200
Questions?