An Improved Analytical Superscalar Microprocessor Memory Model

Xi Chen and Tor Aamodt

Electrical and Computer Engineering
University of British Columbia

June 22nd, 2008 (MoBS-2008)
Outline

- Introduction
- Background
- Modeling Long Latency Memory Systems
  - Pending Hits
  - Accurate Hidden Miss Latency Estimation
  - A Limited Number of Outstanding Cache Misses
  - SWAM & SWAM-MLP
- Methodology
- Results
- Future Work and Conclusions
Introduction

- Processor design is a complicated task

- Cycle-accurate performance simulators are extensively used to explore design space
  - Creating and debugging a simulator takes time
  - Running detailed simulations is slow

Xi Chen, Tor Aamodt
University of British Columbia
Introduction

- Simulating future large-scale CMPs?

Time to simulate 1 second of LCMP (32-core, 4-thread per core) workload execution (Li Zhao et al., Exploring Large-Scale CMP Architectures Using ManySim, IEEE Micro, Issue 4, 2007)
Introduction

- Analytical Modeling
  - an alternative to cycle-accurate simulations

![Diagram showing program characteristics, microarchitectural parameters, Analytical Model, and performance.]
Introduction

- Analytical Modeling vs. Performance Simulations
  - **Pros**
    - Fast speed (orders of magnitude times faster)
    - Providing more insights for chip designers
  - **Cons**
    - Less accurate than performance simulations
    - Covering only major microarchitectural parameters
Background

- **First-order Model**
  - Stable performance is disrupted by different types of miss-events

![Graph showing IPC over time with miss-events]

Xi Chen, Tor Aamodt
University of British Columbia

An Improved Analytical Superscalar Microprocessor Memory Model
Background

First-order Model

- The overall performance (CPI) is modeled by the sum of isolated CPI components due to each type of miss-event.

\[ \text{CPI}_{\text{modeled}} = \text{CPI}_{\text{base}} + \text{CPI}_{\text{bmpsp}} + \text{CPI}_{D$\text{miss}} + \text{CPI}_{I$\text{miss}} \]

- The **baseline** in our paper is our careful re-implementation of the first-order model based upon the available details described in the ISCA’04 paper and its follow-up work.
Pending Data Cache Hits

CPI_Dmiss (mcf) vs. memory access latency (cycle)
Contributions

- **Pending Hits**
  - error is reduced
  - 43.5% -> 27.5%

- **Accurate Hidden Miss Latency Estimation**
  - 15.5%
  - -> 10.3%

- **Profile Window Selection**
  - baseline
  - contribution
  - 29.2%
  - -> 10.3%

  - 32.4%
  - -> 9.2%

---

Xi Chen, Tor Aamodt
University of British Columbia

An Improved Analytical Superscalar Microprocessor Memory Model
Modeling $CPI_{D\text{miss}}$ : Baseline

- Assuming the instruction window size is eight

$$\text{num}_{\text{serialized}}_{D\text{miss}}: 0 \rightarrow 1$$
Modeling CPI\text{D$\text{miss}$ : Baseline

- Assuming the instruction window size is eight

\text{num\_serialized\_D$\text{miss}$: 1 -> 3}
Modeling $CPI_{D:\text{miss}}$ : **Baseline**

$$CPI_{D:\text{miss}} = \frac{\text{num\_serialized\_D:\text{miss} } \times \text{mem\_lat}}{\text{total\_num\_instructions}}$$

- **mem\_lat**: main memory latency
- **total\_num\_instructions**: total number of instructions committed
Pending Data Cache Hits

- A pending data cache hit results from a memory reference to a cache block for which a request has already been initiated by another instruction.

- Pending data cache hits are common due to spatial locality in applications.
Pending Data Cache Hits

- A (traditional) cache simulator cannot identify pending cache hits due to the lack of timing information.
Pending Data Cache Hits

| i1 | ld r2, (0)r1 |
| i2 | ld r3, (4)r1 |
| i3 | add r4, r2, r5 |
| i4 | add r6, r3, r5 |
| i5 | ld r7, (0)r6 |
| i6 | or r8, r4, r5 |
| i7 | add r9, r5, r7 |
| i8 | or r10, r8, r9 |
| i9 | ...... |

not considering pending hits (num_serialized_D$miss += 1)

wrong value
Modeling Pending Hits

| i1 | ld r2, (0)r1 | miss |
| i2 | ld r3, (4)r1 | hit (i1) |
| i3 | add r4, r2, r5 |
| i4 | add r6, r3, r5 |
| i5 | ld r7, (0)r6 | miss |
| i6 | or r8, r4, r5 |
| i7 | add r9, r5, r7 |
| i8 | or r10, r8, r9 |
| i9 | ...... |

considering pending hits (num Serialized_D$miss += 2) correct value

latency modeled by CPI_{base}

Error is reduced from 43.5% to 27.5% with best fixed cycle compensation for miss latency

An Improved Analytical Superscalar Microprocessor Memory Model
Compensating Overestimate
[K&S ISCA’04, Karkhanis PhD Thesis]

Part of the latency of a miss may be hidden when a miss issues, it can be (or near) the commit side of the ROB

\[
CPI_{D\text{miss}} = \frac{\text{num}_{-}\text{serialized}_{-}D\text{miss} \times \text{mem}_{-}\text{lat}}{\text{total}_{-}\text{num}_{-}\text{instructions}}
\]
Part of the latency of a miss may be hidden when a miss issues, it can be (or near) the middle of the ROB.

\[ CPI_{D_{miss}} = \frac{\text{num}_{-}\text{serialized}_{-}D_{miss} \times (\text{mem}_{-}\text{lat} - \frac{\text{ROB}_{size}}{2 \times \text{issue}_{-}\text{width}})}{\text{total}_{-}\text{num}_{-}\text{instructions}} \]
Compensating Overestimate
[K&S ISCA’04, Karkhanis PhD Thesis]

- Part of the latency of a miss may be hidden when a miss issues, it can be (or near) the fetch side of the ROB

\[
CPI_{D\text{miss}} = \frac{\text{num}_{\text{serialized}} \cdot D_{\text{miss}} \times (\text{mem}_{\text{lat}} - \frac{\text{ROB}_{\text{size}}}{\text{issue}_{\text{width}}})}{\text{total}_{\text{num}_{\text{instructions}}}}
\]
Compensating Overestimate

- Fixed-cycle compensation used by prior work is not accurate for all the benchmarks we study.

- We propose to compensate the overestimate using the average distance between consecutive misses.
Compensating Overestimate

over 70% instructions can be used to hide miss latency in pointer chasing

\[
CPI_{D\text{miss}} = \frac{\text{num}_{\text{serialized}} \times \text{D$\text{miss\_penalty} - \text{dist} \times \text{num}_{\text{D$\text{miss}}}}}{\text{total}_{\text{num\_instructions}}}
\]

(dist: average distance between two consecutive misses (the distance between two misses is saturated by the size of ROB))

error is reduced from 15.5% (best fixed cycle compensation) to 10.3%
Modeling a limited number of MSHRs

- The model thus far has assumed that the number of outstanding cache misses supported is unlimited.

- We propose a technique to model a limited number of outstanding cache misses supported.
Modeling a limited number of MSHRs

- We stop a profile step and update `num_serialized_D$miss` when the number of misses analyzed is equal to the number of Miss Status Holding Registers (MSHRs)

\[ \text{ROB}_{\text{size}} = 8, \ N_{\text{MSHR}} = 4 \]

With plain profiling, error reduces from 32.4% to 23.9% for 8 MSHRs
Start-with-a-miss (SWAM) Profiling

- Each profile step starts with a cache miss

Plain profiling (baseline): 29.2% (error of plain) -> 10.3% (error of SWAM)

Sliding profiling: (does not improve the accuracy much but slowdown the model significantly)

SWAM profiling: 29.2% (error of plain) -> 10.3% (error of SWAM)
SWAM-MLP

SWAM (ROB$_{\text{size}}$ = 8, N$_{\text{MSHR}}$ = 4)

num_serialized_miss += 4 (wrong value)

num_serialized_miss += 2 (correct value)

12.8% (error of SWAM) to 9.2% (error of SWAM-MLP) for 8 MSHRs
23.2% (error of SWAM) to 9.9% (error of SWAM-MLP) for 4 MSHRs
Methodology

- We modified SimpleScalar and used a set of memory intensive benchmarks from SPEC 2000 and OLDEN suite whose cache misses per thousand instructions (MPKI) is higher than 10 for our cache configurations.

<table>
<thead>
<tr>
<th>Machine Width</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROB Size</td>
<td>256</td>
</tr>
<tr>
<td>LSQ Size</td>
<td>256</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>4KB gShare</td>
</tr>
<tr>
<td>L1 I-Cache</td>
<td>16KB, 32B/line, 4-way, 1-cycle latency</td>
</tr>
<tr>
<td>L1 D-Cache</td>
<td>16KB, 32B/line, 4-way, 2-cycle latency</td>
</tr>
<tr>
<td>L2 Cache</td>
<td>128KB, 64B/line, 8-way, 10-cycle latency</td>
</tr>
<tr>
<td>Memory Latency</td>
<td>200 cycles</td>
</tr>
</tbody>
</table>
Results

- Unlimited number of outstanding cache misses supported

![Graph showing comparison between Plain w/o PH, Plain w/ PH, and SWAM w/ PH]

- Error (%)

- Application names: app, art, eqk, luc, swm, roc, em, hth, pm, ibm, mean

- Error percentages for each application:
  - app: ~40%
  - art: ~20%
  - eqk: ~60%
  - luc: ~80%
  - swm: ~40%
  - roc: ~0%
  - em: ~-20%
  - hth: ~-40%
  - pm: ~-60%
  - ibm: ~-80%
  - mean: ~39.7%

- Improvement:
  - Plain w/o PH: 39.7%
  - Plain w/ PH: 29.2%
  - SWAM w/ PH: 10.3%
Results

- Modeling a limited number of MSHRs
  - SWAM-MLP further decreases error of SWAM from 12.8% to 9.2% (8 MSHRs, e.g., Prescott), from 23.2% to 9.9% (4 MSHRs)

- Our improvements do not slowdown the first-order model
Current / Future Work

- Analytically modeling the performance impact of hardware data prefetching (by the pending hits caused by data prefetches)

- Analytically modeling the throughput of fine-grain multithreaded microprocessors (e.g., Sun’s Niagara)

- Extending the analytical model for CMPs with superscalar, out-of-order execution cores

- Analytically modeling the performance of SMT superscalar cores
Conclusions

- Modeling Long Latency Memory System
  - Pending Data Cache Hit
  - Accurate Hidden Miss Latency Estimation
  - Modeling MSHRs
  - SWAM & SWAM-MLP

- Overall our improvements reduce the error of our baseline from 39.7% to 10.3%. The error is less than 10% when modeling MSHRs. Our improvements do not slow down the first-order model.