Cross-Architecture Performance Predictions for Scientific Applications Using Parameterized Models

Gabriel Marin
John Mellor-Crummey
Department of Computer Science
Rice University
Los Alamos Computer Science Institute

Performance Modeling Challenges

• Performance depends on:
  —architecture specific factors
  —application characteristics
  —input data parameters

• Difficult to model execution time directly

• Collecting data at scale is expensive

Motivation

• Execution characteristics scaling
• Expected execution time
• Causes of inefficiency
• Estimate payoff of optimizations
• Resource selection at launch-time
• Run-time adaptation
• Future architecture design

Contributions

• Methodology for computing detailed and accurate predictions, including effects from
  —instruction execution
  —memory latency
  —instruction schedule latency

• Scalable performance models
  —predict performance for problem sizes not measured or simulated

• Cross-architecture performance prediction
  —RISC superscalar
  —VLIW
Outline

- New performance modeling technique
  - Capturing dynamic behavior
  - Building scalable models
- Experiments
- Related work
- Conclusions and future work

Approach

- Separate effects of application specific factors on overall performance
- Measure the application-specific factors
  - static analysis
  - dynamic analysis
- Construct scalable models
- Explore interactions with hardware

Toolkit Design Overview

Static Analysis

- Control flow graph
- Loop nesting structure
- Instruction mix
- Instruction dependencies
  - register dependencies
  - memory dependencies
Dynamic Analysis

- Augment program to capture dynamic behavior
- Two measurement modes:
  - block execution frequency + communication characteristics
  - memory reuse distance

Measuring Reuse Distance

MRD: number of distinct memory blocks accessed between two accesses to the same block

<table>
<thead>
<tr>
<th>reference memory block</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRD</td>
<td>I₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reference memory block</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRD</td>
<td>I₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reference memory block</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRD</td>
<td>I₃</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reference memory block</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRD</td>
<td>I₁</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reference memory block</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRD</td>
<td>I₂</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>reference memory block</th>
<th>time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRD</td>
<td>I₃</td>
</tr>
</tbody>
</table>
Measuring Reuse Distance

MRD: number of distinct memory blocks accessed between two accesses to the same block

<table>
<thead>
<tr>
<th>reference memory block</th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRD</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

SIGMETRICS'04 10 June 14, 2004

Measuring Reuse Distance

MRD: number of distinct memory blocks accessed between two accesses to the same block

<table>
<thead>
<tr>
<th>reference memory block</th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
<th>I₁</th>
<th>I₂</th>
<th>I₃</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRD</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
</tbody>
</table>

SIGMETRICS'04 10 June 14, 2004
Measuring Reuse Distance

MRD: number of distinct memory blocks accessed between two accesses to the same block

<table>
<thead>
<tr>
<th>Memory Block Reference</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>I₁</td>
<td>A</td>
</tr>
<tr>
<td>I₂</td>
<td>B</td>
</tr>
<tr>
<td>I₃</td>
<td>C</td>
</tr>
</tbody>
</table>

MRD:

- I₁: 1 cold miss, 1 x distance 1
- I₂: 1 cold miss, 1 x distance 0
- I₃: 1 cold miss, 1 x distance 2

From Data to Models

- Collect data from multiple runs
- Approximation function:
  \[ F(X) = c_n * B_n(X) + c_{n-1} * B_{n-1}(X) + \ldots + c_0 * B_0(X) \]
- Goal: determine coefficients

Execution Frequency Modeling Example

<table>
<thead>
<tr>
<th>X</th>
<th>2</th>
<th>5</th>
<th>9</th>
<th>15</th>
<th>18</th>
<th>23</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>5784</td>
<td>20244</td>
<td>53020</td>
<td>131104</td>
<td>183160</td>
<td>289200</td>
<td></td>
</tr>
</tbody>
</table>

Dynamic Analysis

Static Analysis

Architecture Description

Scheduler

Performance Prediction for Target Architecture

Post Processing Tool

Post Processing
Execution Frequency Modeling Example

<table>
<thead>
<tr>
<th>X</th>
<th>2</th>
<th>5</th>
<th>9</th>
<th>15</th>
<th>18</th>
<th>23</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>Count</td>
<td>5784</td>
<td>20244</td>
<td>53020</td>
<td>131104</td>
<td>183160</td>
<td>289200</td>
<td></td>
</tr>
</tbody>
</table>

**Execution Frequency Model**

- Collected data
- Model degree 0
- Model degree 1

Y = 16776*X - 42366, Err = 60.4%
Y = 41416, Err = 131%

Modeling Memory Reuse Distance

- Memory access behavior is complex
  — cold misses
  — one load instruction ⇒ many reuse distances
- Average reuse distance is misleading
  — 1 access with distance >> cache size
  — 3 accesses with distance < cache size
  ⇒ average distance > cache size
- Model behavior using histograms
  — collect histograms of data reuse distance
  — model the structure and scaling of these histograms
Modeling Memory Reuse Distance

Model constant distance first

Recursively split rest of data

Normalized frequency  Problem size

Normalized frequency  Problem size
Modeling Memory Reuse Distance

Model constant distance first
Recursively split rest of data

Predict Number of Cache Misses

• Instantiate model for problem size 100

Predict Schedule Latency for an Architecture

• Input:
  — basic block and edge execution frequencies

• Methodology:
  — infer executed paths from BB and edge frequencies
  — native instructions → generic RISC instructions
  — instantiate scheduler with architecture description
  — construct instruction schedule for executed paths
    — consider instruction latencies and dependences
Outline

- New performance modeling technique
  - Capturing dynamic behavior
  - Building scalable models

Experiments

- Related work
- Conclusions and future work

Sample Programs

- NAS benchmarks (NPB 2.3 serial and NPB 3.0)
  - SP
  - BT
  - LU (computational fluid dynamics codes)
- ASCI Sweep3D - 3D neutron transport

Predictions of Memory Behavior: NAS BT 3.0

MIPS R12000 (32KB L1, 8MB L2)

Itanium2 (256KB L2, 1.5MB L3)
Predictions of Execution Time: NAS BT 3.0

MIPS R12000 (32KB L1, 8MB L2)

- Measured time
- Scheduler latency
- L1 miss penalty
- L2 miss penalty
- TLB miss penalty
- Predicted time

Predictions of Execution Time: NAS BT 3.0

Itanium2 (256KB L2, 1.5MB L3)

- Measured time
- Scheduler latency
- L2 miss penalty
- L3 miss penalty
- TLB miss penalty
- Predicted time

Related Work

- Reuse distance
  — Cache utilization [Beyls & D’Hollander]
  — Investigating optimizations [Ding et al.]

- Program instrumentation
  — EEL, QPT [Ball, Larus, Schnarr]

- Simulation (trace-based and execution-driven)

- Cross-architecture models at scale
  — [Saavedra & Smith; Snavely et al.; Cascaval et al.]

- Scalable analytical models
  — [Vernon et al; Hoisie et al.]

None yield semi-automatically derived scalable models of node performance

Conclusions & Open Issues

- New performance analysis technique
  — isolate interactions with hardware
  — scalable predictions
  — cross-architecture & cross-compiler predictions
  — semi-automatic

- Open Issues
  — better modeling of memory subsystem
    - # outstanding loads to accurately predict memory latency
  — explore modeling of irregular applications
  — model parallel applications