Automatically Characterizing Large Scale Program Behavior

Timothy Sherwood
Erez Perelman
Greg Hamerly
Brad Calder
Title

- **Ideal**: To understand the effects of cycle-level events on full program execution

- **Challenge**: To achieve this without doing complete detailed simulation

- **How**: Build a high-level model of program behavior that can be used in conjunction with limited detailed simulation
Goals

• The goals of this research are:
  – To create an automatic system that is capable of intelligently characterizing time-varying program behavior
  – To provide both analytic and software tools to help with program phase identification
  – To demonstrate the utility of these tools for finding places to simulate (SimPoints)
  – Without full program detailed simulation
Our Approach

• Programs are neither
  – Completely Homogenous
  – nor Totally Random
• Instead they are quite structured
• Discover this structure

• The key is the code that is executing
  – the code determines the program behavior
Large Scale Behavior (gzip)
Some Definitions

• Interval is
  – A set of instructions that execute one after the other in program order
  – 100 Million Instructions

• Phase is
  – A set of intervals with very similar behavior
  – Regardless of temporal adjacency
Outline

- Examining the Programs
- Finding Phases Automatically
- Application to Efficient Simulation
- Conclusions
Fingerprinting Intervals

- Fingerprint each interval in program
  - Enabling us to build high level model
- Basic Block Vector [PACT’01]
  - Tracks the code that is executing
  - Long sparse vector
  - 1 dimension per static basic block
  - Based on instruction execution frequency
**Basic Block Vectors**

<table>
<thead>
<tr>
<th>BB</th>
<th>Assembly Code of bzip</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>srl a2, 0x8, t4</td>
</tr>
<tr>
<td></td>
<td>and a2, 0xff, t12</td>
</tr>
<tr>
<td></td>
<td>addl zero, t12, s6</td>
</tr>
<tr>
<td></td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0x25, v0</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0, t0</td>
</tr>
<tr>
<td></td>
<td>bis v0, t0, v0</td>
</tr>
<tr>
<td></td>
<td>bne v0, 0x120018c48</td>
</tr>
<tr>
<td>2</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>cmple t7, 0x3, t2</td>
</tr>
<tr>
<td></td>
<td>beq t2, 0x120018b04</td>
</tr>
<tr>
<td>3</td>
<td>ble t7, 0x120018bb4</td>
</tr>
<tr>
<td>4</td>
<td>and t4, 0xff, t5</td>
</tr>
<tr>
<td></td>
<td>srl t4, 0x8, t4</td>
</tr>
<tr>
<td></td>
<td>addl zero, t5, s6</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0x25, s0</td>
</tr>
<tr>
<td></td>
<td>cmpeq s6, 0, a0</td>
</tr>
<tr>
<td></td>
<td>bis s0, a0, s0</td>
</tr>
<tr>
<td></td>
<td>bne s0, 0x120018c48</td>
</tr>
<tr>
<td>5</td>
<td>subl t7, 0x1, t7</td>
</tr>
<tr>
<td></td>
<td>gt t7, 0x120018b90</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

For each interval:

<table>
<thead>
<tr>
<th>ID:</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>BB Exec Count:</td>
<td>&lt;1, 20, 0, 5, 0, ...&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>weigh by Block Size:</td>
<td>&lt;8, 3, 1, 7, 2, ...&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Normalize to 1 =</td>
<td>&lt;8%, 58%, 0%, 34%, 0%, ...&gt;</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- One BBV for each interval
- We can now compare vectors
- Start with simple manual analysis
  - Compare all $N^2$ pairs of intervals
- Enter the Similarity Matrix...
Similarity Matrix

- Compare \( N^2 \) intervals
- Executed Instructions on Diagonal axis
- To compare 2 points go horizontal from one and vertically from the other
- Darker points indicate similar vectors
- Clearly shows the phase-behavior
A More Complex Matrix - gcc

- Still much structure
- Dark boxes show phase-behavior
- Boxes in interior show recurring phases
  Strong diagonal line indicates first half is similar to second half
- Manual inspection is not feasible or scalable
Outline

• Examining the Programs

• Finding Phases Automatically

• Application to Efficient Simulation

• Conclusions
Finding the Phases

• Basic Block Vector is a point in space
• The problem is to find groups of vectors/points that are all similar
  – Making sure that all points in a group are similar to one another
  – And ensuring all points that are different, are put into different groups
• This is a Clustering Problem
• A Phase is a Cluster of BBVectors
Phase-finding Algorithm

I. Profile Program and track BB Vectors

II. Use the K-means algorithm to find clusters in the data for many different values of K

III. Score the likelihood of each clustering

IV. Pick the best clustering
Improving Performance

• K-means requires many manipulations
  – Basic Block Vectors are very long
    • > 100,000 for gcc; 800,000 for microsoft apps
  – Need to make the Vectors smaller
    • Still preserve relative distances

• Random Projection
  – Multiply the vector by a random matrix
  – Can safely reduce down to 15 dimensions
  – Reduce run-time from days to minutes
Example: gzip Revisited

L2
Energy
DL1
IL1
bpred
IPC
gzip – Phases Discovered

L2
Energy
DL1
IL1
bpred
IPC
gcc - A Complex Example

L2
Energy
DL1
IL1
bpred
IPC
gcc – Phases Discovered

L2
Energy
DL1
IL1
bpred
IPC

10B  20B  30B  40B
Outline

- Examining the Programs
- Finding Phases Automatically
  - Application to Efficient Simulation
- Conclusions
Efficient Simulation

• Simulating to completion not feasible
  – Detailed simulation on SPEC takes months
  – Cycle level effects can’t be ignored
• To reduce simulation time
  – Simulate only a subset of the program at cycle-level accuracy
  – What subset you pick is very important
    • For accuracy and efficiency
Simulation Options

• **Simulate Blind**: no estimate of accuracy
• **Single Point**: problem with complex programs that have many phases
• **Random Sample**: high accuracy, but many sections of similar code, you will be doing a lot of redundant work
• **Choose Multiple Points**: by examining the calculated phase information
Multiple SimPoints

• Perform phase analysis

• For each phase in the program
  – Pick the interval most representative of the phase
  – This is the SimPoint for that phase

• Perform detailed simulation for SimPoints

• Weigh results for each SimPoint
  – According to the size of the phase it represents
Results - Average Error

- No FastFwd: 253%
- FastFwd Billion: 131%
- Single SimPoint: 5%
- Multiple SimPoints: 0%
Results – Max Error

- No FastFwd: 3736%
- FastFwd Billion: 1986%
- Single SimPoint: 0%
- Multiple SimPoints: 20%

IPC - Max Error
Outline

• Examining the Programs

• Finding Phases Automatically

• Application to Efficient Simulation

• Conclusions
Conclusions

• Gap between
  – Cycle level events
  – Full program effects

• Exploit large scale structure
  – Provide high level model
  – Find the model with no detail simulation
  – In conjunction with limited detail simulation
Conclusions

• Our Strategy
  – Take advantage of structure found in program
  – Summarize the structure in the form of phases
  – Find phases using techniques from clustering

• Use this for doing efficient simulation
  – High accuracy
  – With orders of magnitude less time

• http://www.cs.ucsd.edu/~sherwood