Performance Cloning: A Technique for Disseminating Proprietary Applications as Benchmarks

Ajay Joshi (University of Texas)
Lieven Eeckhout (Ghent University, Belgium)
Robert H. Bell Jr. (IBM Corp.)
Lizy John (University of Texas)

IEEE International Symposium on Workload Characterization
October 26, 2006
Benchmark Spectrum

- Toy Benchmarks (e.g. Hanoi, Heapsort)
- Microbenchmarks (e.g. STREAM)
- Kernel Codes (e.g. Livermore Loops)
- Synthetic Benchmarks (e.g. Dhrystone, Whetstone)
- Application Suites (e.g. SPEC CPU)
- Complete Application Code

Less Development Effort vs. More Development Effort
More Scalable vs. Less Scalable
More Maintainable vs. Less Maintainable
Less Representative vs. More Representative
Real World Applications as Benchmarks

- Increases confidence in making design tradeoffs
- Customize microprocessor design to specific applications
- Best way to understand processor’s use
- Perhaps the only way to understand emerging workload characteristics ...
- Simplifies purchasing decisions for customers
Challenges With Using Real World Applications

• Real world applications tend to be proprietary
• Using real world applications for performance studies can be tedious
  - Difficult to duplicate user environment
  - Modifying application to research environment
  - Duplicating real input data set
• Real world workloads are a moving target..
The Problem ....

• Need a methodology to create benchmarks that capture the main performance of real world applications

• Resulting benchmarks should hide functional meaning of code

• Ability to study “what-if” scenarios by varying program characteristics
Outline

- Background and Motivation
- Performance Cloning – Central Idea
- Performance Cloning Framework
- Workload Profiling
- Algorithm for Clone Generation
- Analysis and Results
- Summary
Performance Cloning – Central Idea

Real World Application

Measure Inherent Workload Characteristics

Instruction Mix
Basic Block Size
ILP
Data Locality
.............

Workload Characteristics

Performance Clone

Generate Clone with Similar Characteristics

ADD R1, R2, R3
LD R4, R1, R6
MUL R3, R6, R7
ADD R3, R2, R5
DIV R10, R2, R1
SUB R3, R5, R6
STORE R3, R10, R20
ADD R1, R2, R3
LD R4, R1, R6
MUL R3, R6, R7
ADD R3, R2, R5
DIV R10, R2, R1
SUB R3, R5, R1
BEQ R3, R6, LOOP
SUB R3, R5, R6
STORE R3, R10, R20
DIV R10, R2, R1
.............
Performance Cloning Framework

Microarchitecture-Independent Workload Profiling

Modeling Workload Attributes into Synthetic Workload

Experiment Environment

Real World Proprietary Workload

Workload Profiler

*Binary Instrumentation OR Simulation*

Workload Profile = Workload Attributes + Distribution Of Attribute Values

Workload Synthesizer

Synthetic Benchmark Clone

Real Hardware

Execution Driven Simulator
Outline

- Background and Motivation
- Performance Cloning – Central Idea
- Performance Cloning Framework
- Workload Profiling
- Algorithm for Clone Generation
- Analysis and Results
- Summary
Microarchitecture-Independent Profile

• Control Flow Behavior
• Data Locality
• Control Flow Predictability
• Instruction Mix
• Instruction Level Parallelism
Control Flow Behavior (1)

Statistical Flow Graph (K=0)
[Eeckhout et al., ISCA 2004]
Control Flow Behavior (2)

Statistical Flow Graph (K=1)
[EEeckhout et al., ISCA 2004]

Application Binary

Profiling
Modeling Memory Data Access Pattern

- Identify streams of data references

- A Stream?
  - Sequence of memory addresses in an arithmetic progression
  - Elements of arrays A, B, and C form 3 streams

\[
\begin{align*}
\text{for}( \ ii = 0; \ ii < N; \ ii \ ++) \\
A[\ ii] &= B[\ ii] + C[\ ii] \\
& \quad 200, 204, 208 .. \quad 320, 324, 328 .. \quad 404, 408, 412 .. \\
\text{Issuing Sequence:} \quad & 320, 404, 200, 324, 408, 204 .. 
\end{align*}
\]

- Streams are interleaved and may contain noise

  4, 8, 12, 16, 1, 3, 20, 24, 5, 7, 2, 9, 11, 28 ..
Extracting Streams

• Reference pattern of static Load / Store Instructions
  – PC-correlated spatial locality
    - Dependence on address referenced by nearby Ld / St
    - Programs with pointer chasing codes
  – PC-correlated temporal locality
    - Dependence on previous address generated by same Ld / St
    - Programs with multidimensional arrays

• Could static Load / Store instructions be natural sources of streams?

• Profile every static Load / Store instruction
  – Number of different strides with which it accesses data
As a First-Order Model, Static Load/Stores can be modeled as single stream
Modeling Control Flow Predictability

- Capture behavior of easy and difficult to predict branches
- Inherent program feature that captures branch behavior
- Transition Rate [Haungs et al. HPCA’00]
  
  \[
  \text{Transition Rate} = \frac{\text{# of Taken-Not Taken transitions}}{\text{# of times executed}}
  \]

- Branches with low transition-rate (easier to predict)
  \[
  TTTTTTTTTTN, NNNNNNNNT
  \]

- Branches with high transition-rate (easier to predict)
  \[
  TNTNTNTNTN
  \]

- Branches with moderate transition-rate (tougher to predict)
Modeling Instruction Level Parallelism

Dependency Distance

- ADD R1, R3, R4
- MUL R5, R3, R2
- ADD R5, R3, R6
- LD R4, (R8)
- SUB R8, R2, R1

Read After Write Dependency Distance = 3

Measure Distribution of Dependency Distances

- Upto 1
- Upto 2
- Upto 4
- Upto 8
- Upto 16
- Upto 32
- >32
Outline

- Background and Motivation
- Performance Cloning – Central Idea
- Performance Cloning Framework
- Workload Profiling
- Algorithm for Clone Generation
- Analysis and Results
- Summary
Performance Clone Generation

Instruction Mix
Register Dependency Distance
Stride Pattern of Load/Store
Branch Transition Rate
Branch Transition Probabilities

Workload Profile
Performance Clone Generation
Performance Clone Generation

Instruction Mix
Register Dependency Distance
Stride Pattern of Load/Store
Branch Transition Rate
Branch Transition Probabilities

Workload Profile

1 Big Loop

Synthetic Clone Generation

A
B
D
A
B
D
C
D
A
B
D

1.0
0.8
0.2
0.9
0.1
1.0
0.2
0.8
0.2
0.9
0.1
Performance Clone Generation

Instruction Mix
Register Dependency Distance
Stride Pattern of Load/Store
Branch Transition Rate
Branch Transition Probabilities

Workload Profile

Memory Access Model (Strides)

Synthetic Clone Generation

1 Big Loop
Performance Clone Generation

Workload Profile

Instruction Mix
Register Dependency Distance
Stride Pattern of Load/Store
Branch Transition Rate
Branch Transition Probabilities

Memory Access Model (Strides)

Synthetic Clone Generation

1 Big Loop

Branching Model – Based on Transition Rate
Performance Clone Generation

Instruction Mix
Register Dependency Distance
Stride Pattern of Load/Store
Branch Transition Rate
Branch Transition Probabilities

Workload Profile

Memory Access Model (Strides)

1 Big Loop

Synthetic Clone Generation

Branching Model – Based on Transition Rate

Register Assignment
C code with asm & volatile constructs
Outline

- Background and Motivation
- Performance Cloning – Central Idea
- Performance Cloning Framework
- Workload Profiling
- Algorithm for Clone Generation
- Analysis and Results
- Summary
## Tools & Benchmarks

- SimpleScalar/Wattch Simulators for profiling and cycle-accurate simulation
- Alpha ISA – Programs compiled with **Compaq cc v6.3-025 –O3 level**
- Benchmarks from MiBench and MediaBench benchmark suites as representatives of characteristics of Embedded Applications

<table>
<thead>
<tr>
<th>Program</th>
<th>Application Domain</th>
</tr>
</thead>
<tbody>
<tr>
<td>basicmath, qsort, bitcount, susan</td>
<td>Automotive</td>
</tr>
<tr>
<td>crc32, dijkstra, patricia</td>
<td>Networking</td>
</tr>
<tr>
<td>fft, gsm</td>
<td>Telecommunication</td>
</tr>
<tr>
<td>ghostscript, rsynth, stringsearch</td>
<td>Office</td>
</tr>
<tr>
<td>jpeg, typeset</td>
<td>Consumer</td>
</tr>
<tr>
<td>cjpegb, djpegb, g721-decode, ghostscript, mpeg, rasta, rawaudio, texgen, unepic</td>
<td>Media</td>
</tr>
</tbody>
</table>
Evaluation

• Absolute accuracy
  - Ability of performance clone to estimate absolute IPC and Power

• Relative accuracy
  - Sensitivity (IPC and Power) of performance clone to cache & microarchitecture design changes

• Base Configuration

<table>
<thead>
<tr>
<th>Component</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 I-cache</td>
<td>16 KB/2-way/32 B</td>
</tr>
<tr>
<td>L1 D-cache</td>
<td>16 KB/2-way/32 B</td>
</tr>
<tr>
<td>L2 Unified cache</td>
<td>64 KB/4-way/64 B</td>
</tr>
<tr>
<td>Fetch, Decode, and Issue Width</td>
<td>1-wide out-of-order</td>
</tr>
<tr>
<td>Fetch Queue</td>
<td>8 entry</td>
</tr>
<tr>
<td>Branch Predictor</td>
<td>2-level GAp predictor</td>
</tr>
<tr>
<td>Functional Units</td>
<td>2 Integer ALU, 1 FP Multiplication Unit, 1 FP ALU</td>
</tr>
<tr>
<td>Reorder Buffer</td>
<td>16 entries</td>
</tr>
<tr>
<td>Load Store Queue</td>
<td>8 entries</td>
</tr>
<tr>
<td>Memory (Bus Width, First Block Latency)</td>
<td>8 B, 40 cycles</td>
</tr>
</tbody>
</table>
Absolute Accuracy in IPC

Average absolute error in estimating IPC is 8.7%
Absolute Accuracy in Power

Average absolute error in estimating power is 6.4%
Tracking Design Changes (1)

Across 28 cache configurations
Tracking Design Changes (2)

Across 28 cache configurations
## Tracking Design Changes (3)

<table>
<thead>
<tr>
<th>Design Change</th>
<th>Average Relative Error in IPC</th>
<th>Average Relative Error in Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double the number of entries in the reorder buffer and load store Queue</td>
<td>5.81%</td>
<td>3.41%</td>
</tr>
<tr>
<td>Reduce the L1 cache size to half</td>
<td>1.48%</td>
<td>0.39%</td>
</tr>
<tr>
<td>Double the fetch, decode, and issue Width</td>
<td>5.41%</td>
<td>4.59%</td>
</tr>
<tr>
<td>Change the predictor from a 2-level to a not-taken predictor</td>
<td>6.51%</td>
<td>1.80%</td>
</tr>
<tr>
<td>Change the instruction issue policy to in-order</td>
<td>3.26%</td>
<td>1.22%</td>
</tr>
</tbody>
</table>

5 Different Microarchitecture Changes
Outline

- Background and Motivation
- Performance Cloning – Central Idea
- Performance Cloning Framework
- Workload Profiling
- Algorithm for Clone Generation
- Analysis and Results
- Summary
Conclusions

- Technique that clones performance but hides functional meaning of code
  - Architects & Designers can get access to proprietary workloads
  - Foster benchmark sharing between industry and academia
  - Customers can make informed purchase decisions

- Evaluation of technique on embedded benchmarks is promising
  - Synthetic clone exhibits similar power/performance characteristics
  - Synthetic clone is a good proxy to original application
Challenges & Limitations

• Compiler technology is absorbed into the performance clone
  - Limited use for compiler studies

• Benchmark contains ISA specific embedded asm statements
  - Every embedded microprocessor designer cares about single ISA
  - Possibilities for true portability – virtual ISA, binary translation

• Abstract workload model simple by construction
  - Ability to perform “what-if” performance studies
  - Higher order models to capture complex dataflow
ARE THERE ANY QUESTIONS?