Machine Learning for Performance and Power Modeling/Prediction

Lizy K. John
University of Texas at Austin
Simulation Challenges

- Simulation Based Performance Models
  - eg: SimOS, SIMICS, GEM5, SimpleScalar

- Power modeling
  - eg: McPAT, CACTI

- Full system Simulation is prohibitively slow

- Simulation errors

- Large Gap between what’s evaluated pre-silicon and what’s run post-silicon
Three Examples for Using Machine Learning in Performance/Power Modeling and Prediction

- Calibration of Power Models using Machine Learning
- Cross-Platform Performance/Power Prediction using Machine Learning
- Stressmarks and Power Viruses using Machine Learning
Example 1

Machine Learning for Model Calibration

ISLPED 2015
Example 1

Machine Learning for Model Calibration

Training

Estimated Total Power
- Estimated Power

Measured Total Power
- Measured Power

Correction vector

Prediction

Calibrated Power

Estimated Block Power
Correction Factors – Additive and Multiplicativ

\[ \hat{p}_m = \alpha_G \times (\alpha_0 + \vec{m}_s \times \vec{a}_s + \vec{m}_d \times \vec{a}_d) \]

Non-negative Least Square Error Solver

\[
\text{minimize } \| M \vec{a} - \vec{P} \|^2 + \lambda \| \vec{a} \|, \\
\text{subject to } \alpha \geq 0, \quad \lambda = 0.01
\]
Example 1
Training Process

(a) Odroid Board

(b) DS-5
## Example 1

### Training Process

<table>
<thead>
<tr>
<th>Performance Counter Event</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU_CYCLES</td>
<td>Cycle</td>
</tr>
<tr>
<td>INST RETIRED</td>
<td>Inst architecturally executed</td>
</tr>
<tr>
<td>INST_SPEC</td>
<td>Inst spec exec</td>
</tr>
<tr>
<td>LD_SPEC</td>
<td>Inst spec exec, load</td>
</tr>
<tr>
<td>ST_SPEC</td>
<td>Inst spec exec, store</td>
</tr>
<tr>
<td>ASE_SPEC</td>
<td>Inst spec exec, Advanced SIMD</td>
</tr>
<tr>
<td>VFP_SPEC</td>
<td>Inst spec exec, floating-point</td>
</tr>
<tr>
<td>DP_SPEC</td>
<td>Inst spec exec, int data processing</td>
</tr>
<tr>
<td>BR_IMMED_SPEC</td>
<td>Branch spec exec, immediate branch</td>
</tr>
<tr>
<td>BR_RETURN_SPEC</td>
<td>Branch spec exec, procedure return</td>
</tr>
<tr>
<td>BR_INDIRECT_SPEC</td>
<td>Branch spec exec, indirect branch</td>
</tr>
<tr>
<td>BR_MIS_PRED</td>
<td>Mis- or not predicted branch spec</td>
</tr>
<tr>
<td>L1I_CACHE</td>
<td>Level 1 instruction cache access</td>
</tr>
<tr>
<td>L1I_CACHE_REFILL</td>
<td>Level 1 instruction cache refill</td>
</tr>
<tr>
<td>L1I_TLB_REFILL</td>
<td>Level 1 instruction TLB refill</td>
</tr>
<tr>
<td>L1D_CACHE_LD</td>
<td>Level 1 data cache access, read</td>
</tr>
<tr>
<td>L1D_CACHE_ST</td>
<td>Level 1 data cache access, store</td>
</tr>
<tr>
<td>L1D_CACHE_REFILL_LD</td>
<td>Level 1 data cache refill, read</td>
</tr>
<tr>
<td>L1D_CACHE_REFILL_ST</td>
<td>Level 1 data cache refill, write</td>
</tr>
<tr>
<td>L1D_CACHE_WB</td>
<td>Level 1 data cache write-back</td>
</tr>
<tr>
<td>L1D_TLB_REFILL</td>
<td>Level 1 data TLB refill</td>
</tr>
<tr>
<td>L2D_CACHE_LD</td>
<td>Level 2 data cache access, read</td>
</tr>
<tr>
<td>L2D_CACHE_ST</td>
<td>Level 2 data cache access, store</td>
</tr>
<tr>
<td>L2D_CACHE_REFILL_LD</td>
<td>Level 2 data cache refill, read</td>
</tr>
<tr>
<td>L2D_CACHE_REFILL_ST</td>
<td>Level 2 data cache refill, write</td>
</tr>
<tr>
<td>L2D_CACHE_WB</td>
<td>Level 2 data cache write-back</td>
</tr>
</tbody>
</table>
Example 1

Machine Learning for Model Calibration
Example 1

Calibrated Power

![Calibrated Power Graphs for MB1, MB2, MB3](Image)

- MB1
- MB2
- MB3

- X-axis: Time (ms)
- Y-axis: Power (W)

Calibrated Power Graphs for MB1, MB2, and MB3 showing the power consumption over time with different markers for different scenarios.
Example 2

Machine Learning for Cross-Platform Prediction

**Motivation:**
Full System Simulation is too slow.

Analytical Models are not accurate enough.

Bridge the gap between the two using machine learning.

**Intuition:**
Performance on two platforms is correlated.

Can machine learning be used to understand that correlation?
Example 2


Constrained LASSO Regression
Use Cases for Cross-Platform Prediction

Slow Simulator – No time to run all benchmarks but fast previous generation or other ISA hardware available - Run some benchmarks and use machine learning

Limited Access to New Hardware – Make some runs – Train using them and predict power of other benchmarks

Hardware Software Co-development - If they can run code on existing hardware and predict based on the cross-platform model provided by hardware developer
Learning Formulation

For each application $i$ in the training set we have,

- $x_i \in \mathbb{R}^d$: $d$-dimensional feature vector obtained from the host
- $y_i \in \mathbb{R}$: reference performance on the target

We want to find a function $\mathcal{F}$ such that,

$$\mathcal{F}(x_i) \approx y_i$$

Prediction models:

- Assume locally linear $\mathcal{F}_t$ at input $x_t$ ($\mathcal{F}_t(x_t) = w_t^T x_t$)
- Solve constrained LASSO regression around a neighborhood of $x_t$
Training Set – ACM Programming Contest

<table>
<thead>
<tr>
<th>Application Domains</th>
<th>Number of Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation</td>
<td>14</td>
</tr>
<tr>
<td>Enumeration</td>
<td>16</td>
</tr>
<tr>
<td>String Manipulation</td>
<td>30</td>
</tr>
<tr>
<td>Graph Algorithm</td>
<td>26</td>
</tr>
<tr>
<td>Dynamic Programming</td>
<td>21</td>
</tr>
<tr>
<td>Geometry</td>
<td>25</td>
</tr>
<tr>
<td>Recursion</td>
<td>13</td>
</tr>
<tr>
<td>Miscellaneous</td>
<td>12</td>
</tr>
</tbody>
</table>
## Example 2

### Profiling for Training

<table>
<thead>
<tr>
<th></th>
<th>Intel Core i7-920</th>
<th>AMD Phenom II X6</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 Total Cache Misses</td>
<td>L1 Total Cache Misses</td>
<td></td>
</tr>
<tr>
<td>L2 Total Cache Misses</td>
<td>L2 Total Cache Misses</td>
<td></td>
</tr>
<tr>
<td>L3 Total Cache Misses</td>
<td>Branch Misses</td>
<td></td>
</tr>
<tr>
<td>TLB Loads</td>
<td>Instructions</td>
<td></td>
</tr>
<tr>
<td>Unconditional Branches</td>
<td>Cycle Stalled</td>
<td></td>
</tr>
<tr>
<td>Conditional Branches</td>
<td>Cycles</td>
<td></td>
</tr>
<tr>
<td>Branch Misses</td>
<td>L1 Total Cache Accesses</td>
<td>Floating Point Operations</td>
</tr>
<tr>
<td>Instructions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycle Stalled</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Cycles</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L1 Total Cache Accesses</td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2 Total Cache Accesses</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 2

Performance Prediction Accuracy

![Graph showing total cycles and prediction error for different benchmarks.](image)
Example 2

Power Prediction - Accuracy

![Graph showing average power and prediction error for various benchmarks.](image)
Example 2

Prediction at Fine-grain

Per-Phase Cycles (millions)

Basic Blocks (millions)

- Red: Predicted Cycles Based on Intel Core i7
- Green: Predicted Cycles Based on AMD Phenom II
- Blue: Cycles Measured from Hardware

Lizy K. John 5/12/17
Power Prediction at Fine-grain

Example 2

- Predicted Power Based on Intel Core i7
- Predicted Power Based on AMD Phenom II
- Power Measured from Hardware
Example 2

Average Cross-Validation Error: 10-fold cross-validation

![3D graph showing the relationship between neighborhood size, regularization parameter, and cross-validation error.](image)
Example 2

Non-Linearity of F

![Graph showing Non-Linearity of F with bar chart comparing LASSO and PCLSLR for different error rates and CPU models.](image-url)
Principal components (PC2 vs. PC3) of ICPC training applications (●), MiBench (■) and SD-VBS (♦) programs (Intel Core i7).
Challenges in Cross-Platform Prediction

Host Sensitivity

Instrumenting Source Method

Aligning in No-Source (Perf Counter based) Methodology

Stochastic Dynamic Coupling

Requires Solving Regression during Prediction

More work to be done but cross-platform prediction seems feasible.
Example 3

Challenges in Creating Max-power Viruses

- **Hand crafting code snippets for power viruses**
  - Very tedious process, complex interactions inside the processor
  - Cannot be sure if it is the maximum case
  - Heavily architecture dependent; heavy domain knowledge

- **Automatically generate power viruses**

<table>
<thead>
<tr>
<th>Power Virus</th>
<th>Optimized to stress</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPrime</td>
<td>All CPUs, all ISAs</td>
<td>C</td>
</tr>
<tr>
<td>CPUburn-in</td>
<td>X86 machines</td>
<td>x86 assembly</td>
</tr>
<tr>
<td>BurnP5</td>
<td>Intel Pentium w&amp;w/o MMX processors</td>
<td>x86 assembly</td>
</tr>
<tr>
<td>BurnP6</td>
<td>Intel PentiumPro, Pentium II, Pentium III and Celeron CPUs</td>
<td>x86 assembly</td>
</tr>
<tr>
<td>BurnK6</td>
<td>AMD K6 processors</td>
<td>x86 assembly</td>
</tr>
<tr>
<td>BurnK7</td>
<td>AMD Athlon/Duron processors</td>
<td>x86 assembly</td>
</tr>
<tr>
<td>BurnMMX</td>
<td>cache/memory interfaces on all CPUs with MMX</td>
<td>x86 assembly</td>
</tr>
</tbody>
</table>
Example 3

Power measurement of Viruses on Hardware

- **BurnK7** – 72.1 Watts

- **SPEC CPU2006**: 416.gamess and 453.povray consume highest power of 63.1 and 59.6 Watts
Example 3

Power Proxies and Viruses using Machine Learning

- Power estimates
- Fitness values 1, 2, … n

Power/Perf Simulator

Machine Learning

Abstract workload specs

Code Generator

Synth 1
Synth 2
…
Synth n

Power Virus
Proxy Workload Generation

- Derive proxy applications from a set of workload characterizations
- Proxies convey no proprietary information, but capture the execution behavior of developer’s applications
- Proxy applications have similar power and performance characteristics as original

Original Workloads

Proxies are miniature and can be run on RTL

Power can be modeled on RTL without OS and without software stack

Performance/Power Clones
Example 3

Automatic Synthetic Benchmark Generation

Application Behavior Space

Instruction Mix

Program Locality

Control Flow Behavior

Thread Level Parallelism

Communication characteristics

Data Sharing Patterns

‘Knobs’ for Changing Program Characteristics

Workload Synthesis Algorithm

Benchmark Synthesizer

Multithreaded Synthetic Benchmark

Compile and Execute

Hardware

Pre-silicon Model
Example 3

Power Virus Generation using Machine Learning

Machine Learning

Power estimates
Fitness values 1,2 … n

Power/Perf Simulator

Synth 1
Synth 2
…
Synth n

Abstract workload specs

Code Generator

Synth 1
Synth 2
…
Synth n

Power Virus
SYMPO and MAMPO Frameworks

- Automatically search for power viruses using an abstract workload model and machine learning

- GA: search heuristic to solve optimization problems

- Choose a random population, evaluate fitness, apply GA operators to generate next population

- Evolve until required fitness achieved
Example 3

SYMPO Framework – Genetic Algorithm

- Individuals -> synthetic workloads,
- Fitness function -> power on the design under study
- Mutation rate, reproduction rate, crossover rate
Example 3

SYMPO Vs Mprime on SPARC ISA

Config 1 - 14% more

Config 2 - 24% more

Config 3 - 41% more
Comparison to SPEC CPU2006 on SPARC ISA

- Comparison to SPEC CPU2006: 74.4Watts compared to 89.8Watts in SYMPO
Comparison to SPEC CPU2006: 111.8 Watts compared to 89.2 Watts, where theoretical maximum is 220 Watts
The auto-generated stressmark (SYMPO) could beat the hand-tuned burnk7
Summary

Machine Learning Techniques can be used to improve Power Modeling and Prediction.

Cross-Platform Prediction using Machine Learning can accurately track performance and power at phase level.

Synthetic Stressmarks created using Genetic Algorithms can excel hand-generated stressmarks.
Thank You! Questions?

Laboratory for Computer Architecture (LCA)
The University of Texas at Austin
lca.ece.utexas.edu

Lizy K. John