Inferred Models for Dynamic and Sparse Hardware-Software Spaces

Weidan Wu, Benjamin C. Lee
Duke University
Trends in Management & Diversity

• Increasingly Sophisticated Management
  – Allocate resources, schedule applications, …
  – Understand HW-SW interactions

• Increasingly Diverse HW & SW
  – Heterogeneous cores, VMs, contention, …
  – Diverse clients, jobs, tasks, …
Mapping Software to Hardware

Heterogeneous HW  \rightarrow  \text{N}
Diverse SW

– Management space explosion (M x N)

A HW-SW Mapping
Profilers Support Management

- But profile sparsity increases with diversity
Inference with Sparse Profiles

HW \rightarrow System Management Space

\leftarrow SW
Outline

• Introduction
• **Inferred Performance Models**
• Generalized Models
• Specialized Models
• Conclusions
Inferred Performance Models

– Models, predictions support management

HW \rightarrow System Management Space

Model
Integrated HW & SW Analysis

- Lays a foundation for run-time management
- Increases diversity among sparse samples
- Prior work separates HW & SW
New Challenges

• Larger space, greater sparsity
  – Data re-usability is critical
  – 30 parameters $\rightarrow$ $5\times10^{15}$ points

• Less structured training data
  – SW profiles from arbitrary, real shards
  – HW profiles from defined, simulated design space
Principles and Strategies

• Enhance data re-usability
  – Shard-level profiles
  – Portable characteristics (μ-arch independent)

• Automate modeling
  – Genetic algorithm
  – Mitigate space explosion
Shard-level Profiles

• Shards: short dynamic instruction segments
• Re-use data among applications
  – New shards resemble existing ones
  – Monolithic profiles only useful when entire application resembles existing one

Profiled Applications

App 1

App 2

App 3

App 4

A Shard

New Application
Shard-level Profiles

• Shards are sparse, randomly sampled segments of 10M instructions

• Shards from diverse applications complement each other, reducing profiling costs

• Shards expose intra-application diversity
Portable Characteristics

- Re-use data among microarchitectures
  - Microarchitecture-independent measures
  - Ex: instruction mix versus cache miss rate
  - Existing SW profiles relevant for new HW
Sharing Supports Inference

• Shards enhances data re-use across SW

• Portability enhances data re-use across HW

• Inferred models require less training data due to enhanced re-use
\[ Y = X^T \times \beta + \epsilon \]

- CPI, ALUs, cache size, … mem instr freq, regression coefficients
- \[
\begin{bmatrix}
1.21 \\
0.89 \\
\vdots \\
2.36 \\
0.71
\end{bmatrix}
\begin{bmatrix}
2 & 128\ldots & 0.39 \\
4 & 64\ldots & 0.27 \\
\vdots & \vdots & \vdots \\
6 & 256\ldots & 0.36
\end{bmatrix}
\begin{bmatrix}
\beta_1 \\
\beta_2 \\
\vdots \\
\beta_p
\end{bmatrix}
\]

- \text{X includes non-linear kernel transformations}
  - Ex: log(cache size)
- \text{X includes pair-wise interactions}
  - Ex: ALU instructions, units
Space of Model Specifications

• Many kernel transformations
  – log, power, cubic spline, exponential, sqrt…
  – 30 parameters, 5 kernels → $5^{30}$ model specs

• Many parameter interactions
  – Hardware and software interact
  – \( \binom{30}{2} = 435 \) pairwise interactions → $2^{435}$ specs
Automatic Model Construction

- Model specification encoded as genes
- Mutation, crossover search models
- Selection evolves model toward higher accuracy
Automatic Model Updates

- New data updates model specification
- Algorithm changes kernels, interactions, fit

New Training Data

Old Model

Genetic Algorithm

Model (kernels, interactions)
Outline

• Introduction
• Inferred Performance Models
• Generalized Models
• Specialized Models
• Conclusions
Generalized Models

• Diverse SW as applications enter/leave system
  – Ex: democratized datacenter computing

• Heterogeneous HW as architectures tuned
  – Ex: big/small cores, VMs, contention, …

• Profiled data collected as SW runs on HW

• Models update to accommodate dynamics
Inductive Hypothesis

– System in steady state
– Accurate model is trained $M(H,S)$
– Manager uses model predictions
Inductive Step

- System is perturbed with new SW or HW
- Profile new SW-HW, check prediction
Model Updates

• Poor prediction triggers model update
  – Collect a few profiles for new SW (e.g., 10-20)
  – Update kernels, interactions, fit
Integrated HW & SW Space

• Hardware Space (17 parameters)
  – Pipeline parameters ➔ e.g. width, rob size
  – Cache parameters ➔ e.g., cache size, associativity
  – Functional unit ➔ e.g., ALU count

• Software Space (13 parameters)
  – Instruction mix
  – Locality ➔ e.g., re-use distance
  – ILP ➔ e.g., producer-consumer distance
Steady State Interpolation

- Train model with sparse HW-SW profiles
- Interpolate for HW-SW pairs not profiles
Perturbed Extrapolation

- Train model with sparse HW-SW profiles
- Extrapolate for new SW and new HW

- Predict app $n$ from $n-1$ apps
- Also supports SW variants (compiler opt, data inputs)
Relative Accuracy

– Accurate interpolation, extrapolation
– Correlation coefficient > 0.9
Outline

• Introduction
• Inferred Performance Models
• Generalized Models
• Specialized Models
• Conclusions
Specialized Models

• Generality is expensive
  – Requires many SW characteristics (e.g., 13)

• With domain knowledge, SW behavior expressed at higher level
  – Reduces number of SW characteristics
  – Reduces profiling cost
  – Increases model accuracy
Sparse Matrix-Vector Multiply

\[ A = \begin{pmatrix}
  a_{00} & a_{01} & 0 & 0 \\
  a_{10} & a_{11} & 0 & 0 \\
  0 & 0 & a_{22} & 0 \\
  0 & 0 & a_{32} & a_{33}
\end{pmatrix} \]

\[ b_{\text{value}} = \begin{pmatrix}
  a_{00} & a_{01} & a_{10} & a_{11} & 0 & 0 & a_{14} & a_{15} & a_{22} & 0 & 0 & a_{33} & 0 & 0 & a_{24} & a_{25} & a_{34} & a_{35}
\end{pmatrix} \]

- Compute \( y = Ax + b \) when \( A \) is sparse, blocked
- SW space \( \rightarrow \) block row, block column, fill ratio
- HW space \( \rightarrow \) cache
## SpMV Model Accuracy

- Models irregular performance caused by fill ratios

### Performance Topology (nasasrb hb, Mflop/s)

#### Baseline Arch (Observed)

<table>
<thead>
<tr>
<th>block row</th>
<th>block column 1</th>
<th>block column 2</th>
<th>block column 3</th>
<th>block column 4</th>
<th>block column 5</th>
<th>block column 6</th>
<th>block column 7</th>
<th>block column 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.80</td>
<td>0.94</td>
<td>0.99</td>
<td>0.94</td>
<td>0.94</td>
<td>0.99</td>
<td>1.02</td>
<td>0.85</td>
</tr>
<tr>
<td>7</td>
<td>0.82</td>
<td>0.94</td>
<td>1.01</td>
<td>0.94</td>
<td>0.90</td>
<td>1.02</td>
<td>0.85</td>
<td>0.85</td>
</tr>
<tr>
<td>6</td>
<td>1.37</td>
<td>1.43</td>
<td>1.74</td>
<td>1.41</td>
<td>1.18</td>
<td>1.78</td>
<td>1.02</td>
<td>1.02</td>
</tr>
<tr>
<td>5</td>
<td>0.93</td>
<td>1.09</td>
<td>1.17</td>
<td>1.05</td>
<td>0.98</td>
<td>1.18</td>
<td>0.90</td>
<td>0.90</td>
</tr>
<tr>
<td>4</td>
<td>1.06</td>
<td>1.27</td>
<td>1.36</td>
<td>1.19</td>
<td>1.09</td>
<td>1.42</td>
<td>0.95</td>
<td>0.95</td>
</tr>
<tr>
<td>3</td>
<td>1.31</td>
<td>1.56</td>
<td>1.69</td>
<td>1.37</td>
<td>1.18</td>
<td>1.77</td>
<td>1.03</td>
<td>1.01</td>
</tr>
<tr>
<td>2</td>
<td>1.19</td>
<td>1.48</td>
<td>1.62</td>
<td>1.33</td>
<td>1.14</td>
<td>1.73</td>
<td>1.00</td>
<td>1.00</td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.31</td>
<td>1.50</td>
<td>1.23</td>
<td>1.09</td>
<td>1.63</td>
<td>0.97</td>
<td>0.90</td>
</tr>
</tbody>
</table>

#### Baseline Arch (Predicted)

<table>
<thead>
<tr>
<th>block row</th>
<th>block column 1</th>
<th>block column 2</th>
<th>block column 3</th>
<th>block column 4</th>
<th>block column 5</th>
<th>block column 6</th>
<th>block column 7</th>
<th>block column 8</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>0.79</td>
<td>0.89</td>
<td>0.95</td>
<td>0.93</td>
<td>0.86</td>
<td>1.01</td>
<td>0.84</td>
<td>0.87</td>
</tr>
<tr>
<td>7</td>
<td>0.80</td>
<td>0.91</td>
<td>0.97</td>
<td>0.94</td>
<td>0.87</td>
<td>0.98</td>
<td>0.82</td>
<td>0.82</td>
</tr>
<tr>
<td>6</td>
<td>1.29</td>
<td>1.54</td>
<td>1.74</td>
<td>1.35</td>
<td>1.15</td>
<td>1.70</td>
<td>1.00</td>
<td>1.01</td>
</tr>
<tr>
<td>5</td>
<td>0.90</td>
<td>1.11</td>
<td>1.19</td>
<td>1.05</td>
<td>0.91</td>
<td>1.17</td>
<td>0.89</td>
<td>0.87</td>
</tr>
<tr>
<td>4</td>
<td>0.98</td>
<td>1.23</td>
<td>1.37</td>
<td>1.23</td>
<td>1.05</td>
<td>1.39</td>
<td>0.97</td>
<td>0.95</td>
</tr>
<tr>
<td>3</td>
<td>1.25</td>
<td>1.57</td>
<td>1.86</td>
<td>1.42</td>
<td>1.25</td>
<td>1.85</td>
<td>1.05</td>
<td>1.01</td>
</tr>
<tr>
<td>2</td>
<td>1.11</td>
<td>1.43</td>
<td>1.74</td>
<td>1.41</td>
<td>1.28</td>
<td>1.80</td>
<td>1.09</td>
<td>1.05</td>
</tr>
<tr>
<td>1</td>
<td>1.00</td>
<td>1.27</td>
<td>1.59</td>
<td>1.29</td>
<td>1.19</td>
<td>1.74</td>
<td>1.10</td>
<td>1.07</td>
</tr>
</tbody>
</table>

### True performance

### Predictive performance
Also in the paper...

• Shard-level prediction
  – Basis of application prediction

• Genetic algorithm evaluation
  – Convergence versus model accuracy

• Coordinated optimization for SpMV
  – Optimize HW and software
  – Optimize performance and power
Conclusions

• Present framework to close data-to-decision gap
• Infer performance from huge, sparse data
• Automate modeling in dynamic managers
• Apply domain knowledge for concise models
Inferred Models for Dynamic and Sparse Hardware-Software Spaces

Weidan Wu, Benjamin C. Lee
Duke University