Blind Optimization for Exploiting Hardware Features

Dan Knights, Todd Mytkowicz, Peter F. Sweeney*, Michael C. Mozer, and Amer Diwan

University of Colorado at Boulder

*IBM Research, Hawthorne
Motivation: Microprocessors are complex

- Hard to optimize for all hardware features simultaneously
Motivation: Some features hidden

- Not all hardware details are published (e.g. trace cache)
Motivation: Compilers use simplified models

- Predictive heuristics optimize features independently
- Models may fail to capture interactions
The variant space

program → compiler

\[ \begin{align*}
\text{binary } \alpha \\
\text{binary } \beta \\
\text{binary } \gamma \\
\text{binary } \delta \\
\text{binary } \epsilon \\
\text{binary } \zeta \\
\text{binary } \eta \\
\end{align*} \]

\{ program variants \}
Our variant space: Function alignments

- Why function alignments?
  - Can affect multiple processor features (e.g. caches, branch prediction, LSD)
  - Standard heuristics are clearly insufficient

<table>
<thead>
<tr>
<th>variant β</th>
<th>quantum_toffoli()</th>
<th>+57</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>quantum_sigma_x()</td>
<td>+8</td>
</tr>
<tr>
<td></td>
<td>quantum_cnot()</td>
<td>+15</td>
</tr>
</tbody>
</table>

64-byte boundaries
Challenge: which variant is fastest?

<table>
<thead>
<tr>
<th>Variant α</th>
<th>Variant β</th>
</tr>
</thead>
<tbody>
<tr>
<td>+0 quantum_toffoli()</td>
<td>+57 quantum_toffoli()</td>
</tr>
<tr>
<td>+0 quantum_sigma_x()</td>
<td>+8 quantum_sigma_x()</td>
</tr>
<tr>
<td>+0 quantum_cnot()</td>
<td>+15 quantum_cnot()</td>
</tr>
</tbody>
</table>

64-byte boundaries
Challenge: which variant is fastest?

- **Variant $\alpha$**
  - Quantum Toffoli: $2.6 \times 10^9$ cycles
  - Quantum Sigma X: $+57$ cycles
  - Quantum CNOT: $+15$ cycles

- **Variant $\beta$**
  - Quantum Toffoli: $2.3 \times 10^9$ cycles
  - Quantum Sigma X: $+8$ cycles
  - Quantum CNOT: $+$ cycles

64-byte boundaries

GCC -O3
Results: distribution of run-times

Benchmark hmmer

GCC on all inputs, all benchmarks: 57th percentile
The problem: a huge variant space

- variant $\alpha$
- variant $\varepsilon$
- variant $\beta$
- variant $\gamma$
- variant $\delta$

program variant space
The good news: There is room for improvement

- Variant $\alpha$
- Variant $\varepsilon$
- Variant $\beta$
- Variant $\gamma$
- Variant $\delta$

Computer

Program variant space
Blind Optimization: Oblivious to hardware details

variant

quantum_toffoli()

quantum_sigma_x()

quantum_cnot()

Input

Computer

Runtime

Dan Knights (CU Boulder)
Blind Optimization
Approach:
Do a series of random experiments

variant α

quantum_toffoli()
quantum_sigma_x()
quantum_cnot()

Input

Computer

+
Approach:
Do a series of random experiments

variant $\beta$

- quantum_toffoli()
- quantum_sigma_x()
- quantum_cnot()

Input

Computer

Output
Approach:
Do a series of random experiments

variant $\gamma$

- `quantum_toffoli()`
- `quantum_sigma_x()`
- `quantum_cnot()`

+ Input

Computer

Clipboard
Approach:
Do a series of random experiments

variant $\delta$

- `quantum_toffoli()`
- `quantum_sigma_x()`
- `quantum_cnot()`

Computer

Input

---

Dan Knights (CU Boulder) Blind Optimization
### Approach: Setting up the problem

<table>
<thead>
<tr>
<th></th>
<th>Alignment mod 64</th>
<th>cycles x 10^6</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>GCC -O3:</strong></td>
<td>0, 0, 0,</td>
<td>?</td>
</tr>
<tr>
<td><strong>variant α:</strong></td>
<td>2, 31, 40,</td>
<td>?</td>
</tr>
<tr>
<td><strong>variant β:</strong></td>
<td>57, 8, 15,</td>
<td>?</td>
</tr>
</tbody>
</table>
Approach: Supervised Learning

Given:

\[
\begin{align*}
\begin{bmatrix}
  x_{0,0} & x_{0,1} & \cdots & x_{0,d} \\
  x_{1,0} & x_{1,1} & \cdots & x_{1,d} \\
  \vdots & \vdots & \ddots & \vdots \\
  x_{n,0} & x_{n,1} & \cdots & x_{n,d}
\end{bmatrix}
& \quad \text{d function alignments} \\
& \quad \text{observed runtime} = f(\mathbf{x})
\end{align*}
\]

Find approximation of \( \hat{f}(\mathbf{x}) \), predict \( y' = \hat{f}(\mathbf{x}') \) for a new \( \mathbf{x}' \)
Approach: Direct optimization

- Instead of finding $\hat{f}(\hat{x})$
- Find $\min_{\hat{x} \in X} f(\hat{x})$
  - random search
  - hill-climbing
  - genetic algorithms
  - simulated annealing
  - beam search
  - etc.

\[
\begin{array}{c|c}
\hat{x}_0 & f(\hat{x}_0) = 2.3 \times 10^9 \\
\hat{x}_1 & f(\hat{x}_1) = 2.6 \times 10^9 \\
\hat{x}_2 & f(\hat{x}_2) = 2.4 \times 10^9 \\
\hat{x}_3 & f(\hat{x}_3) = 1.9 \times 10^9 \\
\vdots & \vdots \\
\hat{x}_n & f(\hat{x}_n) = 2.2 \times 10^9 \\
\end{array}
\]
Methodology

• For each benchmark:
  1) Hold out one “test” input
  2) Time 100 random variants on the other inputs
  3) “Vote” for the best variant
  4) Time the winner on the “test” input
  5) Repeat for each input

• Report the average score (cross-validation)
## Results

<table>
<thead>
<tr>
<th>SPEC CPU2006 Benchmark**</th>
<th>Linear model (% speedup*)</th>
<th>Direct optimization (% speedup*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>libquantum</td>
<td>13.2%</td>
<td>13.4%</td>
</tr>
<tr>
<td>lbm</td>
<td>1.6</td>
<td>1.7</td>
</tr>
<tr>
<td>hmemmer</td>
<td>1.5</td>
<td>1.4</td>
</tr>
<tr>
<td>bzip2</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>sjeng</td>
<td>-0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>gobmk</td>
<td>-1.6</td>
<td>0.6</td>
</tr>
<tr>
<td>h264ref</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>mcf</td>
<td>-0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>perlbench</td>
<td>0.0</td>
<td>0.3</td>
</tr>
<tr>
<td>milc</td>
<td>-1.6</td>
<td>0.1</td>
</tr>
<tr>
<td>gcc</td>
<td>-0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>Average</td>
<td>1.1%</td>
<td>1.8%</td>
</tr>
</tbody>
</table>

*Over GCC -O3  ** Intel Xeon dual core / Core 2 Duo
## Results

<table>
<thead>
<tr>
<th>SPEC CPU2006 Benchmark**</th>
<th>Linear model (% speedup*)</th>
<th>Direct optimization (% speedup*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>libquantum</td>
<td>13.2%</td>
<td>13.4%</td>
</tr>
<tr>
<td>lbm</td>
<td>1.6</td>
<td>1.7</td>
</tr>
<tr>
<td>hmmer</td>
<td>1.5</td>
<td>1.4</td>
</tr>
<tr>
<td>bzip2</td>
<td>0.8</td>
<td>0.9</td>
</tr>
<tr>
<td>sjeng</td>
<td>-0.7</td>
<td>0.6</td>
</tr>
<tr>
<td>gobmk</td>
<td>-1.6</td>
<td>0.6</td>
</tr>
<tr>
<td>h264ref</td>
<td>0.3</td>
<td>0.4</td>
</tr>
<tr>
<td>mcf</td>
<td>-0.8</td>
<td>0.4</td>
</tr>
<tr>
<td>perlbench</td>
<td>0.0</td>
<td>0.3</td>
</tr>
<tr>
<td>milc</td>
<td>-1.6</td>
<td>0.1</td>
</tr>
<tr>
<td>gcc</td>
<td>-0.6</td>
<td>0.1</td>
</tr>
<tr>
<td>Average</td>
<td>1.1%</td>
<td>1.8%</td>
</tr>
</tbody>
</table>

*Over GCC -O3  ** Intel Xeon dual core / Core 2 Duo

4.6% ICC
More good news:
We can run more experiments offline

11 PM: Go to bed.
2 AM: Defrag hard-drive.
3 AM: Reoptimize software.
Checkpoint

- Blind optimizations can work
  - even a simple optimization, minimal search
  - up to 13% speedup
- Bad news: big program variant space
- Good news: potential for improvement
- What about other dimensions?
  - function alignments vs. inverting branch directions
Going further: Possible objective functions

GOOD
No dependencies, Structured

BAD
Dependencies, Structured

WORSE
Dependencies, No structure
Combining optimizations:
Sorted by NMDS and Manhattan distance
Combining optimizations:
Sorted by "default" speedup
Prior work: search-based optimizations

* Uses heuristics
Prior work: search-based optimizations

Program-general

Program-specific

# of Features Optimized

Massalin '87
McGovern et al. '02

Holy Grail

* Uses heuristics
Prior work: search-based optimizations

Program-specific

Program-general

Cavazos et al. '04*, '05*
Singer et al. '07*
Holy Grail etc.

Massalin '87
McGovern et al. '02

* Uses heuristics
Prior work: search-based optimizations

**Program-general**
- Cavazos et al. '04*, '05*
- Singer et al. '07*
- Holy Grail etc.

**Program-specific**
- Massalin '87
- McGovern et al. '02

* Uses heuristics

Iterative Compilation* (many)

# of Features Optimized

0 1 2 4 8 ... ∞
Prior work: search-based optimizations

- Program-general
  - Cavazos et al. '04*, '05*
  - Singer et al. '07*
  - Holy Grail etc.

- Program-specific
  - Massalin '87
  - McGovern et al. '02
  - Blind Optimization

Iterative Compilation*
(many)

* Uses heuristics
Summary

- Blind optimization optimizes directly, without heuristics or static models
  - The search space is huge, but there is potential for improvement
  - There are highly complex interactions between optimizations
- Ongoing work:
  - Exploring other dimensions
  - Improving search