Impact of different compiler options on energy consumption

James Pallister
Research Engineer, Embecosm
PhD Student, University of Bristol
(Supervisor: Kerstin Eder)

Simon Hollis
University of Bristol

Jeremy Bennett
Embecosm
Motivation

- Compiler optimizations are claimed to have a large impact on software:
  - Performance
  - Energy
- No extensive study prior to this considering:
  - Different benchmarks
  - Many individual optimizations
  - Different platforms
- This work looks at the effect of many different optimizations across 10 benchmarks and 5 platforms.
- Over 200 optimization passes covered by 150 flags
  - Huge amount of combinations
This Talk

- This talk will cover:
  - Importance of benchmarks
  - How to explore $2^{150}$ combinations of options
  - Correlation between time and energy
  - How to predict the effect of the optimizations
  - The best optimizations
Importance of Benchmarks

• One benchmark can't trigger all optimizations
• Perform differently on different platforms
• Need a range of benchmarks

• Broad categories to be considered for a benchmark:
  – Integer
  – Floating point
  – Branching
  – Memory
Existing Benchmark Suites Considered

- MiBench
- WCET
- DSPstone
- ParMiBench
- OpenBench
- LINPACK
- Livermore Fortran Kernels
- Dhry/Whet-stone

- Require embedded Linux
- Targeted at higher-end systems
- Multithreaded benchmarks typically for HPC
- Don't necessarily test all corners of the platform
## Our Benchmark List

<table>
<thead>
<tr>
<th>Name</th>
<th>Source</th>
<th>B</th>
<th>M</th>
<th>I</th>
<th>FP</th>
<th>T</th>
<th>License</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blowfish</td>
<td>MiBench</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>L</td>
<td>Multi</td>
<td>GPL</td>
<td>security</td>
</tr>
<tr>
<td>CRC32</td>
<td>MiBench</td>
<td>M</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>Single</td>
<td>GPL</td>
<td>network, telecomm</td>
</tr>
<tr>
<td>Cubic root solver</td>
<td>MiBench</td>
<td>L</td>
<td>M</td>
<td>H</td>
<td>L</td>
<td>Single</td>
<td>GPL</td>
<td>automotive</td>
</tr>
<tr>
<td>Dijkstra</td>
<td>MiBench</td>
<td>M</td>
<td>L</td>
<td>H</td>
<td>L</td>
<td>Multi</td>
<td>GPL</td>
<td>network</td>
</tr>
<tr>
<td>FDCT</td>
<td>WCET</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>H</td>
<td>Single</td>
<td>None†</td>
<td>consumer</td>
</tr>
<tr>
<td>Float Matmult</td>
<td>WCET</td>
<td>M</td>
<td>H</td>
<td>M</td>
<td>M</td>
<td>Single</td>
<td>GPL</td>
<td>automotive, consumer</td>
</tr>
<tr>
<td>Integer Matmult</td>
<td>WCET</td>
<td>M</td>
<td>M</td>
<td>H</td>
<td>L</td>
<td>Single</td>
<td>None†</td>
<td>automotive</td>
</tr>
<tr>
<td>Rjindael</td>
<td>MiBench</td>
<td>H</td>
<td>L</td>
<td>M</td>
<td>L</td>
<td>Multi</td>
<td>GPL</td>
<td>security</td>
</tr>
<tr>
<td>SHA</td>
<td>MiBench</td>
<td>H</td>
<td>M</td>
<td>M</td>
<td>L</td>
<td>Multi</td>
<td>GPL</td>
<td>network, security</td>
</tr>
<tr>
<td>2D FIR</td>
<td>WCET</td>
<td>H</td>
<td>M</td>
<td>L</td>
<td>H</td>
<td>Single</td>
<td>None†</td>
<td>automotive, consumer</td>
</tr>
</tbody>
</table>
# Platforms Chosen

<table>
<thead>
<tr>
<th></th>
<th>ARM Cortex-M0</th>
<th>ARM Cortex-M3</th>
<th>ARM Cortex-A8</th>
<th>XMOS L1</th>
<th>Adapteva Epiphany</th>
</tr>
</thead>
<tbody>
<tr>
<td>Memory</td>
<td>Small memory</td>
<td>Small memory</td>
<td>Large memory</td>
<td>Small memory</td>
<td>On-chip and off-chip memory</td>
</tr>
<tr>
<td>Pipeline</td>
<td>Simple Pipeline</td>
<td>Simple Pipeline, with forwarding logic, etc.</td>
<td>Complex superscalar pipeline</td>
<td>Simple pipeline</td>
<td>Simple superscalar pipeline</td>
</tr>
<tr>
<td>SIMD/FPU</td>
<td></td>
<td></td>
<td></td>
<td>SIMD/FPU</td>
<td>FPU</td>
</tr>
<tr>
<td>Multiple threads</td>
<td></td>
<td></td>
<td></td>
<td>Multiple threads</td>
<td>16 cores</td>
</tr>
<tr>
<td>Cores</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Experimental Methodology

- Compiler optimizations have many non-linear interactions
- 238 optimization passes combined into 150 different options (GCC)
- 82 compiler options enabled by O3
- How to test all of these, while accounting for the interactions between optimizations?

Fractional Factorial Designs
Full Factorial Design

Example:
- 3 options to investigate
- Each option can be on or off (2 level)
- $2^3$ tests to be run
Estimating an Option's Effect

- The effect of a single option can be calculated.

\[
X_1 = \frac{\sum}{4} - \frac{\sum}{4}
\]
Fractional Factorial Design

- Use a subset of the full factorial design
- Shown here is a 'half fraction'
- $2^{(3-1)}$ tests to be run
Loss of Information

- Less runs = less information
- The fewer runs performed, the fewer interactions can be resolved
- The 'resolution' of the fractional factorial design

<table>
<thead>
<tr>
<th>Resolution</th>
<th>Runs Needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>256</td>
</tr>
<tr>
<td>4</td>
<td>1024</td>
</tr>
<tr>
<td>5</td>
<td>2048</td>
</tr>
<tr>
<td>6</td>
<td>4096</td>
</tr>
<tr>
<td>Full</td>
<td>137438953472</td>
</tr>
</tbody>
</table>

| 10 hours | 77000 years |
Hardware Measurements

- 10 kSamples/s
- XMOS board to control and timestamp measurements
- Integrate to get energy consumption
Results

- Energy consumption ≈ Execution time
  - Generalization, not true in every case

- Optimization unpredictability

- No optimization is universally good across benchmarks and platforms
Overview

FDCT, Cortex-M0

FDCT, Cortex-A8

Execution time
Overview

FDCT, Cortex-M0

FDCT, Cortex-A8

Execution time

Energy consumed
Overview

FDCT, Cortex-M0

FDCT, Cortex-A8

Legend:
- + Average power
- × Execution time
- ● Energy consumed
Overview
Time = Energy

O1 Flags, Blowfish, Cortex-M0
Time ≈ Energy

- **fpeephole2**
  Constant folding, strength reduction, algebraic simplification

- **fschedule-insns**
  Reorder instructions to reduce execution stalls

- **fcaller-saves**
  “Enable values to be allocated in registers that will be clobbered by function calls, by emitting extra instructions to save and restore the registers around such calls.”

O2 Flags, Blowfish, Cortex-M3
When Time ≠ Energy

- Complex pipeline
- `-ftree-vectorize`
  - NEON SIMD unit
  - Much lower power

O3 Flags, 2DFIR, Cortex-A8
Conclusion: Mostly, Time \approx Energy

- Highly correlated
- Especially so for 'simple' pipelines
- Little scope for stalling or superscalar execution

- Complex pipelines:
  - Still a correlation
  - But more variability
  - SIMD, superscalar execution

- To get the most optimal energy consumption we need better than “go fast”
Optimization Unpredictability

- Pairs of optimizations on top of O0
- Possibly higher order interactions occurring?

O1 Flags, Cubic, Cortex-M0
# The Best Three Optimizations for Energy

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Cortex-M0</th>
<th>Cortex-M3</th>
<th>Cortex-A8</th>
<th>Epiphany</th>
</tr>
</thead>
<tbody>
<tr>
<td>blowfish</td>
<td>B, J, E</td>
<td>J, B, G</td>
<td>K, B, E</td>
<td>D, P, H</td>
</tr>
<tr>
<td>crc32</td>
<td>F</td>
<td>F</td>
<td>F, G</td>
<td></td>
</tr>
<tr>
<td>cubic</td>
<td>A, I</td>
<td>A, I</td>
<td>A</td>
<td>A, I, O</td>
</tr>
<tr>
<td>dijkstra</td>
<td>I, A, B</td>
<td>F, I, A</td>
<td>F, I, A</td>
<td></td>
</tr>
<tr>
<td>float_matmult</td>
<td>C, E</td>
<td>C, E, G</td>
<td>N, L</td>
<td>D, H, A</td>
</tr>
<tr>
<td>int_matmult</td>
<td>C, E, B</td>
<td>C, L, F</td>
<td>L, N, M</td>
<td>A, H, D</td>
</tr>
<tr>
<td>rijndael</td>
<td>B, C, E</td>
<td>C, B, F</td>
<td>C, B, M</td>
<td>D, C, Q</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Count</th>
<th>Flag</th>
<th>ID</th>
<th>Count</th>
<th>Flag</th>
<th>ID</th>
<th>Count</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>11</td>
<td>-ftree-dominator-opts</td>
<td>B</td>
<td>10</td>
<td>-fomit-frame-pointer</td>
<td>C</td>
<td>10</td>
<td>-ftree-loop-optimize</td>
</tr>
<tr>
<td>D</td>
<td>7</td>
<td>-fdce</td>
<td>E</td>
<td>7</td>
<td>-fguess-branch-probability</td>
<td>F</td>
<td>7</td>
<td>-fmove-loop-invariants</td>
</tr>
<tr>
<td>G</td>
<td>7</td>
<td>-ftree-ter</td>
<td>H</td>
<td>6</td>
<td>-ftree-ch</td>
<td>I</td>
<td>6</td>
<td>-ftree-fre</td>
</tr>
<tr>
<td>J</td>
<td>5</td>
<td>-ftree-forwprop</td>
<td>K</td>
<td>4</td>
<td>-fschedule-insns</td>
<td>L</td>
<td>3</td>
<td>-finline-small-functions</td>
</tr>
<tr>
<td>M</td>
<td>3</td>
<td>-fschedule-insns2</td>
<td>N</td>
<td>3</td>
<td>-ftree-pre</td>
<td>O</td>
<td>1</td>
<td>-fcombine-stack-adjustments</td>
</tr>
<tr>
<td>P</td>
<td>1</td>
<td>-fipa-profile</td>
<td>Q</td>
<td>1</td>
<td>-ftree-pta</td>
<td>R</td>
<td>1</td>
<td>-ftree-sra</td>
</tr>
<tr>
<td>S</td>
<td>1</td>
<td>-fgcse</td>
<td>T</td>
<td>1</td>
<td>-fpeephole2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Time $\approx$ Energy
  - True for simple pipelines
  - Mostly true for complex pipelines
  - Good approximation

- Optimization unpredictability
  - Difficult to predict the interactions between optimizations

- Commonality across platforms
  - Instruction set plays a role
  - Common options for the ARM platforms, but not Epiphany
What does this mean?

For the Compiler Writer

- Current optimization levels (O1, O2, etc.) are a good balance between compile time and performance/energy.
- Never completely optimal
- Machine learning
  - MILEPOST
  - Genetic algorithms
  - MAGEEC
- Current optimizations targeted for performance
- Few (if any) optimizations in current compilers are designed to reduce energy consumption
What am I doing now?

- Superoptimization!
  - For energy
- Partially based on *Peephole Superoptimizers*, S. Bansal, 2006
- Find new optimizations for energy efficiency.
More Info

- Academic paper
- Embecosm Blog – http://www.embecosm.com/blog/
  - Superoptimization
  - Benchmarking
  - Compiler optimizations write-up
  - Coming up: Hardware energy measurements
- Dataset Download
  - http://www.cs.bris.ac.uk/Research/Micro/compileroptions.jsp
- Code
  - https://github.com/jpallister/lowpower-benchmarks
  - https://github.com/jpallister/stm32f4-energy-monitor
- UoB Research Projects
  - http://www.cs.bris.ac.uk/Research/Micro/