Post-compiler Software Optimization for Reducing Energy

Eric Schulte ∗ Jonathan Dorn † Stephen Harding ∗
Stephanie Forrest ∗ Westley Weimer †

*Department of Computer Science
University of New Mexico
Albuquerque, NM 87131-0001

†Department of Computer Science
University of Virginia
Charlottesville, VA 22904-4740

March 4, 2014
Introduction

NSA Datacenter in Bluffdale, Utah

65 megawatts

year

(nationalgeographic.com)
Introduction

-0e energy optimization flag

<table>
<thead>
<tr>
<th>Status</th>
<th>RESOLVED INVALID</th>
</tr>
</thead>
<tbody>
<tr>
<td>Product</td>
<td>new-bugs</td>
</tr>
<tr>
<td>Component</td>
<td>new-bugs</td>
</tr>
<tr>
<td>Version</td>
<td>unspecified</td>
</tr>
<tr>
<td>Platform</td>
<td>PC All</td>
</tr>
<tr>
<td>Importance</td>
<td>P normal</td>
</tr>
<tr>
<td>Assigned To</td>
<td>Unassigned LVM Bugs</td>
</tr>
<tr>
<td>URL</td>
<td></td>
</tr>
<tr>
<td>Keywords</td>
<td>new-feature</td>
</tr>
<tr>
<td>Depends on</td>
<td></td>
</tr>
<tr>
<td>Blocks</td>
<td></td>
</tr>
</tbody>
</table>

*Note: You need to log in before you can comment on or make changes to this bug.*

**Description**

Current CPUs like Core i5 or i7 are quite redundant, they can perform the same operations to various ways. Such ways use different amounts of electricity. Then stripped down CPUs like Atom have some redundancy.

In very big data centers, like Google's, the limiting factor is often the amount of electricity used by the very large number of computers present. Mobile phones, ebook readers, and similar gadgets often have powerful CPUs on a chip with many MHz of speed, but for them it's important to maximize the number of hours they can run on a battery charge.

So in theory it can be added a "-0e" compilation argument to LLVM that tells the compiler to compile the code in a way to minimize the amount of electricity the running program will use on the specified CPU.

For example it -0e make the compiler can do the opposite of what I've written in bug-6210, and use the FP stack to store a single float value, saving the power needed to store an move around 32 bits in a SSE register.

**Comment**

Almost all computer literature shows that the best way to save power (on mainstream apps that don't use new crazy power features) is to execute code faster so that the chip can enter a low power state sooner. Faster = lower power.
-Oe energy optimization flag

-0e ... trying to minimize the amount of energy the program will use
-0e energy optimization flag

-0e ... trying to minimize the amount of energy the program will use

Faster = lower energy
Outline

Introduction

Technical Approach

Experimental Evaluation

Conclusion
Problem Statement

Optimizing complex non-functional properties

properties $\times$ hardware $\times$ environment

properties memory, network, energy, etc.

hardware architectures, processors, memory stack, etc.

environment variables, load, etc.

Every program transformation requires

- a-priori reasoning
- manual implementation
- guaranteed correctness
Our Solution

Genetic optimization algorithm

▶ empirically guided (guess and check)
▶ automated evolutionary search
▶ relaxed semantics

Applied to PARSEC benchmarks

▶ reduces energy consumption by 20% on average
▶ maintain functionality on withheld tests
Related Work

Extends combines and leverages

- profile guided optimization
- genetic programming
- superoptimization
- profiling
- testing
Technique

Post-compiler, test-driven, Genetic Optimization Algorithm

Post-compiler

source → GCC → .s → GOA → .s → .exe
Technique

Post-compiler, test-driven, Genetic Optimization Algorithm

Test driven

Use test cases to exercise program

▶ evaluate functionality
▶ measure runtime properties
Technique

Post-compiler, test-driven, **Genetic Optimization Algorithm**

Genetic

```asm
movl $0, -8(%rbp)
.L4:
    addl $1, -4(%rbp)
.L3:
    movl -4(%rbp), %eax
    cmpl -28(%rbp), %eax
    jl .L5
    cmpl $0, -8(%rbp)
    je .L6
```

```asm
movl $0, -8(%rbp)
.L4:
    addl $1, -4(%rbp)
.L3:
    movl -4(%rbp), %eax
    cmpl -28(%rbp), %eax
    jl .L5
    cmpl $0, -8(%rbp)
    je .L6
```
Technique

Post-compiler, test-driven, Genetic Optimization Algorithm

Optimization algorithm

Iteratively improve performance (energy) over time

Swaptions Fitness Over Time

Joules

Iterations
Technique

Post-compiler, test-driven, Genetic Optimization Algorithm

Benefits

▶ environment-specific adaptation
▶ hardware-specific adaptation
▶ exploit hidden HW complexities

(Mdf / CC-BY-SA-3.0)
GOA Overview

Assembler  Fitness Function  Workload
GOA Overview

Assembler → Fitness Function → Workload

Population
GOA Overview

Assembler -> Population

Population -> Mutate

Fitness Function

Workload

Technical Approach for Reducing Energy
GOA Overview

- Assembler
- Fitness Function
- Workload
- Population
- Mutate
- Profile

Technical Approach

Post-compiler Software Optimization for Reducing Energy
GOA Overview

Assembler

Fitness Function

Workload

Population

Fitness

Profile

Mutate

Minimize

Executable

Search

Technical Approach 9

Post-compiler Software Optimization for Reducing Energy
GOA Overview

Assembler
Fitness Function
Workload
Population
Fitness
Profile
Search
Mutate

Technical Approach
Post-compiler Software Optimization for Reducing Energy
GOA Overview

Assembler → Population → Fitness → Profile
Minimize → Population → Mutate → Profile
Fitness Function → Population
Workload → Profile

Search
Post-compiler Software Optimization for Reducing Energy
GOA Overview

Assembler -> Fitness Function

Fitness Function -> Workload

Workload -> Profile

Profile -> Mutate

Mutate -> Population

Population -> Minimize

Minimize -> Executable
Program Mutation

Software Representation

```
movl $0, -8(%rbp)
.L4:
    addl $1, -4(%rbp)
.L3:
    movl -4(%rbp), %eax
    cmpl -28(%rbp), %eax
    jl .L5
    cmpl $0, -8(%rbp)
    je .L6
```

Mutation Operations

- **Copy**
- **Delete**
- **Swap**
- **Two Point Crossover**

Post-compiler Software Optimization for Reducing Energy

Technical Approach
Profiling

Hardware Performance Counters
$ perf stat -- ./blackscholes 1 input /tmp/output

6,864,315,342 cycles
5,062,293,918 instructions
2,944,060,039 r533f00
1,113,084,780 cache-references
1,122,960 cache-misses

3.227585368 seconds time elapsed
Fitness Function

\[
\frac{\text{energy}}{\text{time}} = C_{\text{const}} + C_{\text{ins}} \frac{\text{ins}}{\text{cycle}} + C_{\text{flops}} \frac{\text{flops}}{\text{cycle}} + C_{\text{tca}} \frac{\text{tca}}{\text{cycle}} + C_{\text{mem}} \frac{\text{mem}}{\text{cycle}}
\]
Steady State Genetic Algorithm

Details

- population size: $2^{10}$
- $2^{18}$ fitness evaluations
- $\sim 16$ hour runtime per optimization
Minimization

Delta Debugging

5358c5358
< .L808:
---
> addl %ebx, %ecx

5416c5416
< addl %ebx, %ecx
---
> .L808:

5463c5463
< .L970:
---
> .byte 0x33

5651d5650
< .loc 1 457 0 is_stmt 0 discriminator 2

5841d5839
< addq %rdx, %r14

6309c6307
< xorpd %xmm1, %xmm7
---
> cmpq %r13, %rdi

6413a6412
> cmpl %ecx, %esi
Minimization
Delta Debugging

5358c5358
< .L808:
---
> addl %ebx, %ecx

5416c5416
< addl %ebx, %ecx
---
> .L808:

5463c5463
< .L970:
---
> .byte 0x33

5651d5650
< .loc 1 457 0 is_stmt 0 discriminator 2

5841d5839
< addq %rdx, %r14

6309c6307
< xorpd %xmm1, %xmm7
---
> cmpq %r13, %rdi

6413a6412
> cmpl %ecx, %esi
Minimization

Delta Debugging

5358c5358
< .L808:
  ---
  > addl %ebx, %ecx

5416c5416
< addl %ebx, %ecx
  ---
  > .L808:

5463c5463
< .L970:
  ---
  > .byte 0x33

5651d5650
< .loc 1 457 0 is_stmt 0 discriminator 2

5841d5839
< addq %rdx, %r14

6309c6307
< xorpd %xmm1, %xmm7
  ---
  > cmpq %r13, %rdi

6413a6412
> cmpl %ecx, %esi
Minimization
Delta Debugging

5358c5358
< .L808:
---
> addl %ebx, %ecx

5416c5416
< addl %ebx, %ecx
---
> .L808:

5463c5463
< .L970:
---
> .byte 0x33

5651d5650
< .loc 1 457 0 is_stmt 0 discriminator 2

5841d5839
< addq %rdx, %r14

6309c6307
< xorpd %xmm1, %xmm7
---
> cmpq %r13, %rdi

6413a6412
> cmpl %ecx, %esi
## Benchmark Applications

<table>
<thead>
<tr>
<th>Program</th>
<th>C/C++ Lines of Code</th>
<th>ASM</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>510</td>
<td>7,932</td>
<td>Finance modeling</td>
</tr>
<tr>
<td>bodytrack</td>
<td>14,513</td>
<td>955,888</td>
<td>Human video tracking</td>
</tr>
<tr>
<td>facesim</td>
<td></td>
<td></td>
<td>no alternate inputs</td>
</tr>
<tr>
<td>ferret</td>
<td>15,188</td>
<td>288,981</td>
<td>Image search engine</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>11,424</td>
<td>44,681</td>
<td>Fluid dynamics animation</td>
</tr>
<tr>
<td>freqmine</td>
<td>2,710</td>
<td>104,722</td>
<td>Frequent itemset mining</td>
</tr>
<tr>
<td>raytrace</td>
<td></td>
<td></td>
<td>no testable output</td>
</tr>
<tr>
<td>swaptions</td>
<td>1,649</td>
<td>61,134</td>
<td>Portfolio pricing</td>
</tr>
<tr>
<td>vips</td>
<td>142,019</td>
<td>132,012</td>
<td>Image transformation</td>
</tr>
<tr>
<td>x264</td>
<td>37,454</td>
<td>111,718</td>
<td>MPEG-4 video encoder</td>
</tr>
<tr>
<td>total</td>
<td>225,467</td>
<td>1,707,068</td>
<td></td>
</tr>
</tbody>
</table>

**Table:** PARSEC benchmark applications.
Hardware Platforms

AMD Server

Intel Desktop

Post-compiler Software Optimization for Reducing Energy

Experimental Evaluation
### Energy Model

\[
\frac{\text{energy}}{\text{time}} = C_{\text{const}} + C_{\text{ins}} \frac{\text{ins}}{\text{cycle}} + C_{\text{flops}} \frac{\text{flops}}{\text{cycle}} + C_{\text{tca}} \frac{\text{tca}}{\text{cycle}} + C_{\text{mem}} \frac{\text{mem}}{\text{cycle}}
\]

<table>
<thead>
<tr>
<th>Coefficient</th>
<th>Description</th>
<th>Intel (4-core)</th>
<th>AMD (48-core)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_{\text{const}} )</td>
<td>constant power draw</td>
<td>31.530</td>
<td>394.74</td>
</tr>
<tr>
<td>( C_{\text{ins}} )</td>
<td>instructions</td>
<td>20.490</td>
<td>-83.68</td>
</tr>
<tr>
<td>( C_{\text{flops}} )</td>
<td>floating point ops.</td>
<td>9.838</td>
<td>60.23</td>
</tr>
<tr>
<td>( C_{\text{tca}} )</td>
<td>cache accesses</td>
<td>-4.102</td>
<td>-16.38</td>
</tr>
<tr>
<td>( C_{\text{mem}} )</td>
<td>cache misses</td>
<td>2962.678</td>
<td>-4209.09</td>
</tr>
</tbody>
</table>

**Table:** Energy model coefficients.
Results: Energy Reduction

Energy Reduction

- blackscholes
- bodytrack
- ferret
- fluidanimate
- freqmine
- swaptions
- vips
- x264
- average

AMD
Intel

Post-compiler Software Optimization for Reducing Energy
Experimental Evaluation
Results: Runtime and Energy Reduction

AMD Energy and Runtime Reduction

![Bar chart showing energy and runtime reduction for various applications.](chart.png)
## Functionality on Withheld Tests

<table>
<thead>
<tr>
<th>Program</th>
<th>AMD</th>
<th>Intel</th>
</tr>
</thead>
<tbody>
<tr>
<td>blackscholes</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>bodytrack</td>
<td>92%</td>
<td>100%</td>
</tr>
<tr>
<td>ferret</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>fluidanimate</td>
<td>6%</td>
<td>31%</td>
</tr>
<tr>
<td>freqmine</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>swaptions</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>vips</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>x264</td>
<td>27%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Anecdotes
Blackscholes

- 90% less energy
- removed redundant outer loop
- modified semantics
Anecdotes

Blackscholes

- 90% less energy
- removed redundant outer loop
- modified semantics

Swaptions

- 42% less energy
- improved branch prediction
- hardware specific
Anecdotes

Blackscholes
- 90% less energy
- removed redundant outer loop
- modified semantics

Swaptions
- 42% less energy
- improved branch prediction
- hardware specific

Vips
- 20% less energy
- substitution of memory access for calculation
- resource trade-off
Caveats

Limitations and Generality

- experimental evaluation
  - energy reduction
  - GCC-produced assembler
  - PARSEC benchmarks
- some benchmarks show no improvement
- requires high-quality test cases
- may change program behavior
Conclusion

1. optimize complex runtime properties (energy)
2. leverages particulars of hardware, and environment
3. reveal compiler inefficiencies
4. find efficiencies, e.g., loop elimination
5. transformations presented as ASM diff
Resources

Genetic Optimization Algorithm

GOA tooling
https://github.com/eschulte/goa

reproduce results

Eric Schulte

email eschulte@cs.unm.edu
homepage https://cs.unm.edu/~eschulte
## Genetic Algorithm

### Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Genprog</th>
<th>GOA</th>
</tr>
</thead>
<tbody>
<tr>
<td>population size</td>
<td>40</td>
<td>$2^{10}$</td>
</tr>
<tr>
<td>evaluations</td>
<td>400</td>
<td>$2^{18}$</td>
</tr>
<tr>
<td>selection</td>
<td>fitness proportionate minutes</td>
<td>tournament of 2 hours</td>
</tr>
<tr>
<td>runtime</td>
<td>minutes</td>
<td>hours</td>
</tr>
</tbody>
</table>
Software Mutational Robustness

![Bar chart 1](bubble-sort, insertion-sort, quick-sort, merge-sort)

![Bar chart 2](printtokens, schedule, sed, space, tcas)

![Bar chart 3](C, ASM)

- bzip2 1.0.2
- ccrypt 1.2
- imagemagick 6.5.2
- lighttpd 1.4.15
- nullhttpd 0.5.0
- oggenc 1.0.1
- potion 40b5f03
- redis 1.3.4
- tiff 3.8.2
- vyquon 335426d
- grep
- C
- ASM
Program Syntactic Space

Specifying

(acceptable)

Killed Mutants

Neutral Mutants

Equivalent Mutant

Original Program

Test Suite

Post-compiler Software Optimization for Reducing Energy

Backup 30