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

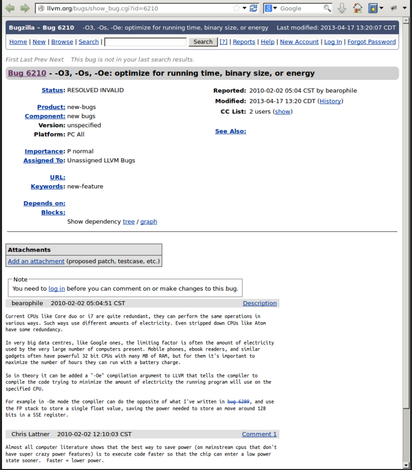
NSA Datacenter in Bluffdale, Utah

$$\frac{65 megawatts}{year}$$

(nationalgeographic.com)

Introduction

-Oe energy optimization flag



llvm.org/bugs/show_bug.cgi?id=6210

Bugzilla – Bug 6210 -O3, -Os, -Oe: optimize for running time, binary size, or energy Last modified: 2013-04-17 13:20:07 CDT

Home | New | Browse | Search | Search | [?] | Reports | Help | New Account | Log In | Forgot Password

First Last Prev Next This bug is not in your last search results.

Bug 6210 - -O3, -Os, -Oe: optimize for running time, binary size, or energy

Status: RESOLVED INVALID

Product: new-bugs
Component: new bugs
Version: unspecified
Platform: PC All

Importance: P normal
Assigned To: Unassigned LLVM Bugs

URL:
Keywords: new-feature

Depends on:
Blocks:
Show dependency tree / graph

Reported: 2010-02-02 05:04 CST by bearophile
Modified: 2013-04-17 13:20 CDT (History)
CC List: 2 users (show)

See Also:

Attachments
Add an attachment (proposed patch, testcase, etc.)

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

bearophile 2010-02-02 05:04:51 CST Description

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

In very big data centres, like Google ones, 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 32 bit CPUs with many MB of RAM, but for them it's important to
maximize the number of hours they can run with a battery charge.

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

For example in -Oe mode the compiler can do the opposite of what I've written in bug 6209, and use
the FP stack to store a single float value, saving the power needed to store an move around 128
bits in a SSE register.
```

Chris Lattner 2010-02-02 12:10:03 CST Comment 1

```
Almost all computer literature shows that the best way to save power (on mainstream cpus that don't
have super crazy power features) is to execute code faster so that the chip can enter a low power
state sooner.  Faster = lower power.
```

Introduction

−Oe energy optimization flag


llvm.org/bugs/show_bug.cgi?id=6210

Bugzilla – Bug 6210 -O3, -Os, -Oe: optimize for running time, binary size, or energy Last modified: 2013-04-17 13:20:07 CDT

Home | New | Browse | Search | Search | [?] | Reports | Help | New Account | Log In | Forgot Password

First Last Prev Next This bug is not in your last search results.

Bug 6210 - -O3, -Os, -Oe: optimize for running time, binary size, or energy

Status: RESOLVED INVALID

Product: new-bugs
Component: new bugs
Version: unspecified
Platform: PC All

Importance: P normal
Assigned To: Unassigned LLVM Bugs

URL:
Keywords: new-feature

Depends on:
Blocks:
Show dependency tree / graph

Reported: 2010-02-02 05:04 CST by bearophile
Modified: 2013-04-17 13:20 CDT (History)
CC List: 2 users (show)

See Also:

Attachments
Add an attachment (proposed patch, testcase, etc.)

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

bearophile 2010-02-02 05:04:51 CST Description

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

In very big data centres, like Google ones, 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 32 bit CPUs with many MB of RAM, but for them it's important to maximize the number of hours they can run with a battery charge.

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

For example in -Oe mode the compiler can do the opposite of what I've written in ~~bug 6209~~, and use the FP stack to store a single float value, saving the power needed to store an move around 128 bits in a SSE register.

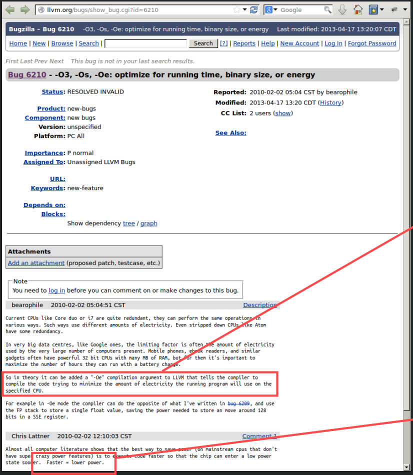
Chris Lattner 2010-02-02 12:10:03 CST Comment 1

Almost all computer literature shows that the best way to save power (on mainstream cpus that don't have super crazy power features) is to execute code faster so that the chip can enter a low power state sooner. Faster = lower power.

−Oe . . . trying to minimize the amount of energy the program will use

Introduction

−Oe energy optimization flag



llvm.org/bugs/show_bug.cgi?id=6210

Bugzilla – Bug 6210 -O3, -Os, -Oe: optimize for running time, binary size, or energy Last modified: 2013-04-17 13:20:07 CDT

Home | New | Browse | Search | Search | [?] | Reports | Help | New Account | Log In | Forgot Password

First Last Prev Next This bug is not in your last search results.

Bug 6210 - -O3, -Os, -Oe: optimize for running time, binary size, or energy

Status: RESOLVED INVALID

Product: new-bugs
Component: new bugs
Version: unspecified
Platform: PC All

Importance: P normal
Assigned To: Unassigned LLVM Bugs

URL:
Keywords: new-feature

Depends on:
Blocks:
Show dependency tree / graph

Reported: 2010-02-02 05:04 CST by bearophile
Modified: 2013-04-17 13:20 CDT (History)
CC List: 2 users (show)

See Also:

Attachments
Add an attachment (proposed patch, testcase, etc.)

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

bearophile 2010-02-02 05:04:51 CST Description

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

In very big data centres, like Google ones, 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 32 bit CPUs with many MB of RAM, but for them it's important to
maximize the number of hours they can run with a battery charge.

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

For example in -Oe mode the compiler can do the opposite of what I've written in bug 6209, and use
the FP stack to store a single float value, saving the power needed to store an move around 128
bits in a SSE register.
```

Chris Lattner 2010-02-02 12:10:03 CST Comment 1

```
Almost all computer literature shows that the best way to save power (on mainstream cpus that don't
have super crazy power features) is to execute code faster so that the chip can enter a low power
state sooner.  Faster = lower power.
```

−Oe . . . trying to minimize the amount of energy the program will use

Faster = lower energy

Outline

Problem Statement

Optimizing complex non-functional properties

$$\textit{properties} \times \textit{hardware} \times \textit{environment}$$

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

```
   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
```

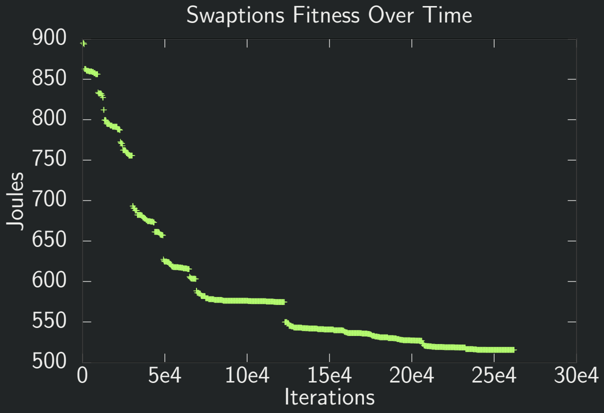
```
 movl  $0, -8(%rbp)
.L4:
 addl  $1, -4(%rbp)
.L3:
 movl  -4(%rbp), %eax
 cmpl  -28(%rbp), %eax
 jl   .L5
```

Technique

Post-compiler, test-driven, Genetic Optimization Algorithm

Optimization algorithm

Iteratively improve performance (energy) over time



Technique

Post-compiler, test-driven, Genetic Optimization Algorithm

Benefits

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

Outline

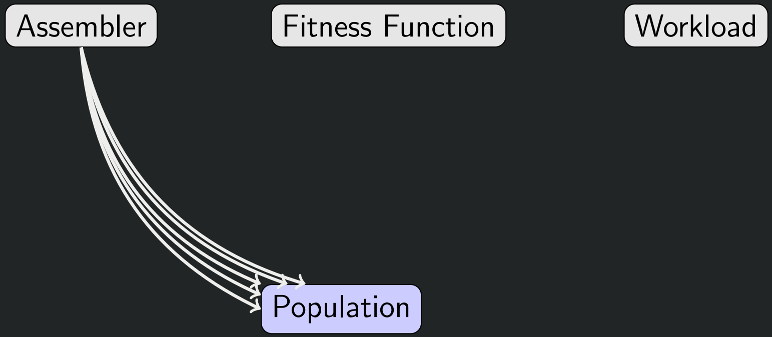
GOA Overview

Assembler

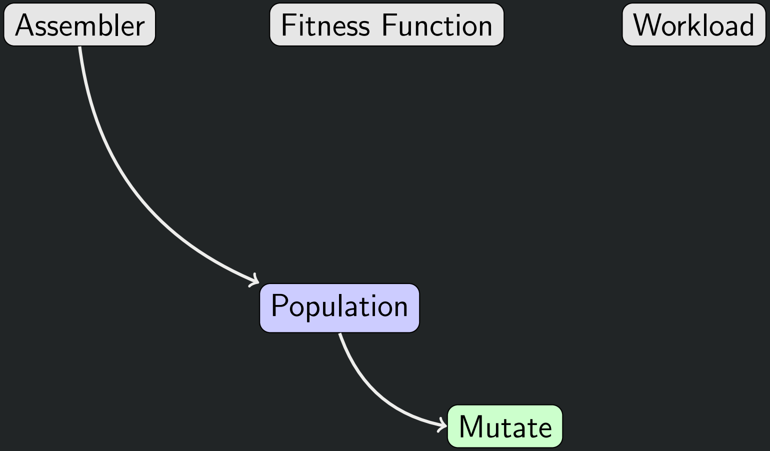
Fitness Function

Workload

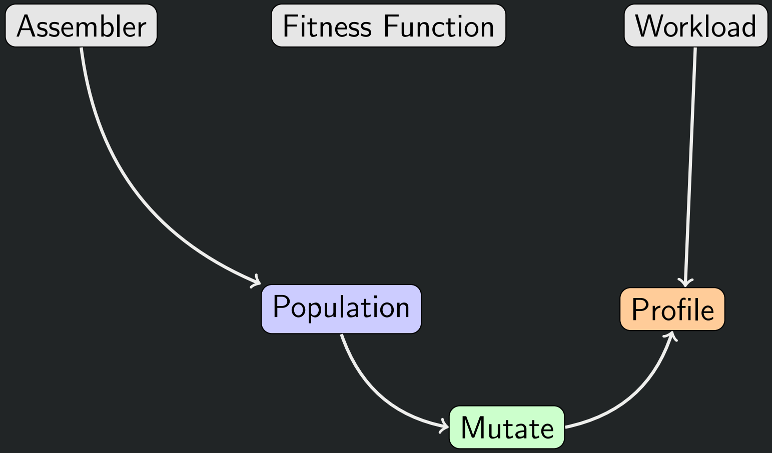
GOA Overview



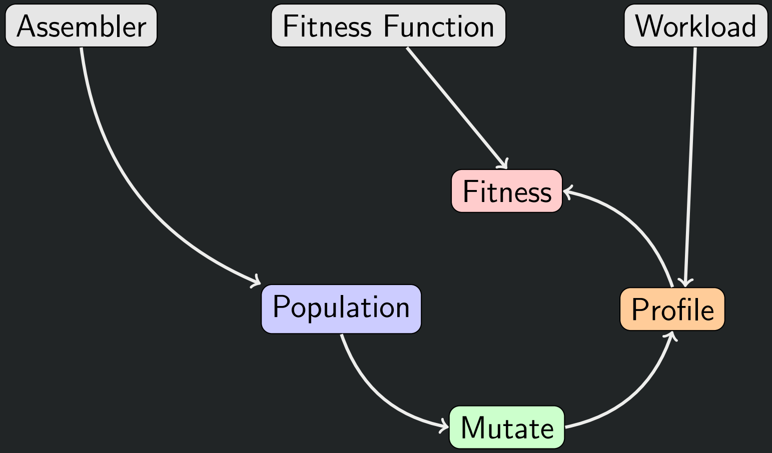
GOA Overview



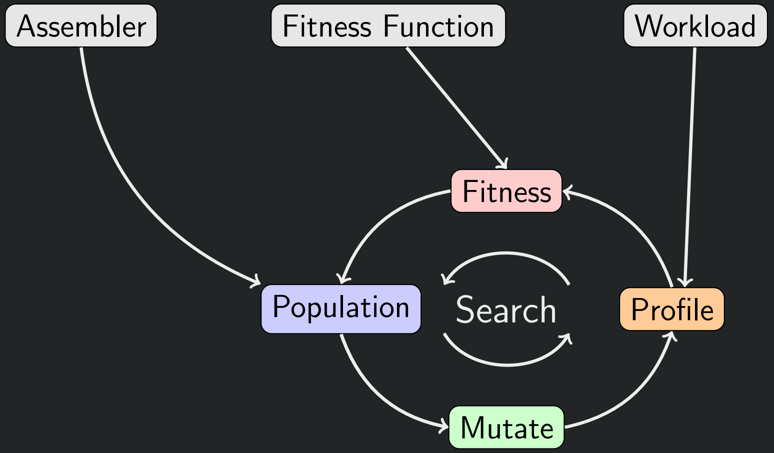
GOA Overview



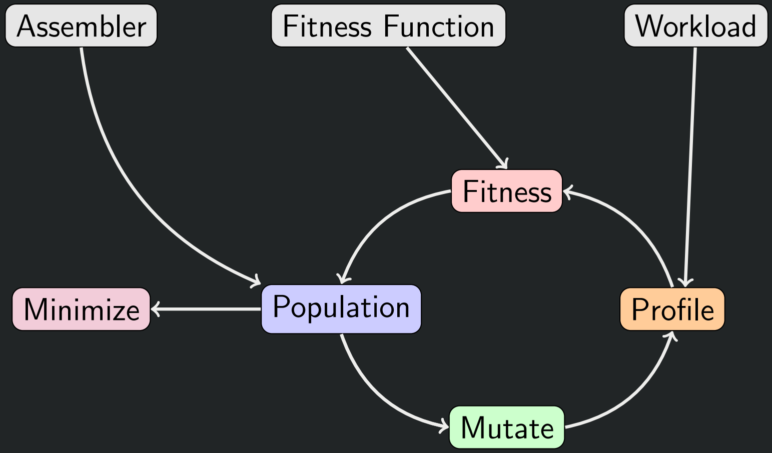
GOA Overview



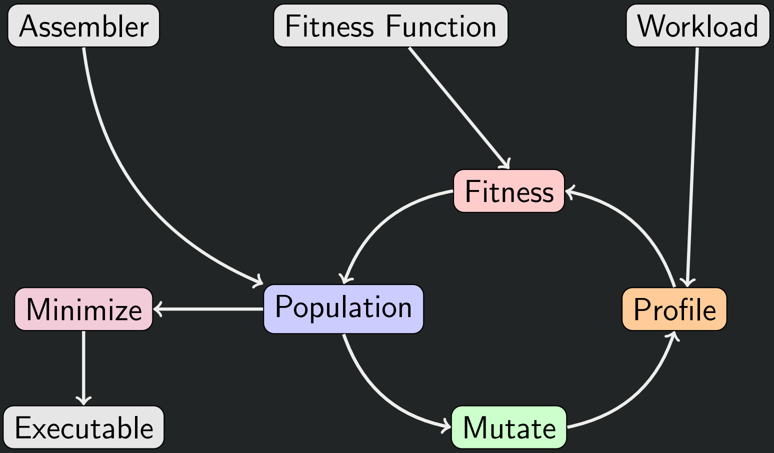
GOA Overview



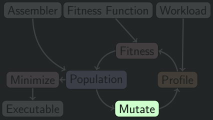
GOA Overview



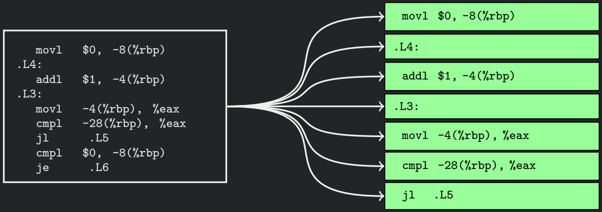
GOA Overview



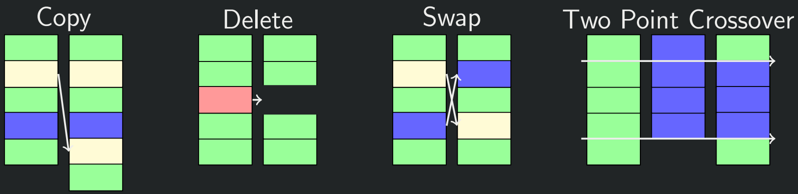
Program Mutation



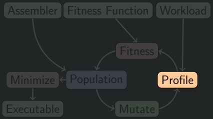
Software Representation

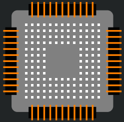


Mutation Operations



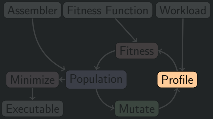
Profiling





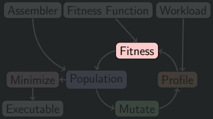
Hardware Performance Counters

Profiling



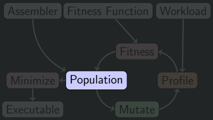

```
$ perf stat -- ./blackscholes 1 input /tmp/output

  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



$$\frac{energy}{time} = C_{const} + C_{ins}\frac{ins}{cycle} + C_{flops}\frac{flops}{cycle} + C_{tca}\frac{tca}{cycle} + C_{mem}\frac{mem}{cycle}$$

Steady State Genetic Algorithm

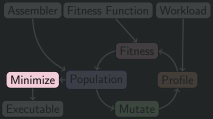


Details

- 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
```

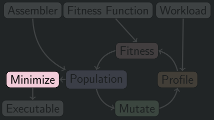
```
5841d5839
<  addq %rdx, %r14
```

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

```
6413a6412
>  cmpl %ecx, %esi
```

Minimization

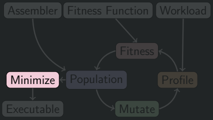
Delta Debugging




```
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

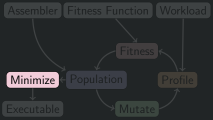
Delta Debugging




```
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
```

Outline

Benchmark Applications

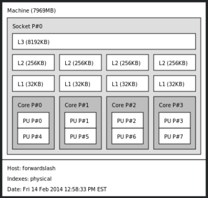
Program	C/C++ Lines of Code	ASM Lines of Code	Description
blackscholes	510	7,932	Finance modeling
bodytrack	14,513	955,888	Human video tracking
facesim			no alternate inputs
ferret	15,188	288,981	Image search engine
fluidanimate	11,424	44,681	Fluid dynamics animation
freqmine	2,710	104,722	Frequent itemset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

Table : PARSEC benchmark applications.

Hardware Platforms

AMD Server

Intel Desktop



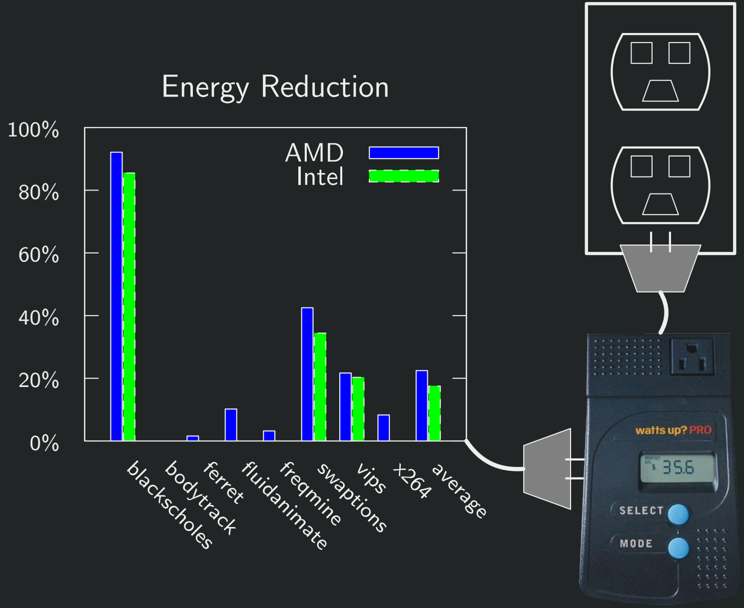
Energy Model

$$\frac{energy}{time} = C_{const} + C_{ins}\frac{ins}{cycle} + C_{flops}\frac{flops}{cycle} + C_{tca}\frac{tca}{cycle} + C_{mem}\frac{mem}{cycle}$$

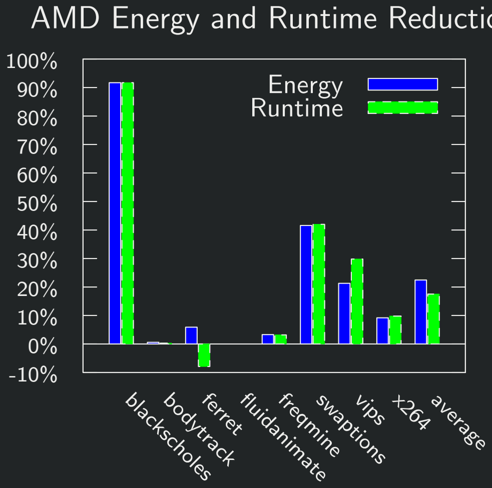
Coefficient	Description	Intel (4-core)	AMD (48-core)
C_{const}	constant power draw	31.530	394.74
C_{ins}	instructions	20.490	-83.68
C_{flops}	floating point ops.	9.838	60.23
C_{tca}	cache accesses	-4.102	-16.38
C_{mem}	cache misses	2962.678	-4209.09

Table : Energy model coefficients.

Results: Energy Reduction



Results: Runtime and Energy Reduction



Functionality on Withheld Tests

Program	AMD	Intel
blackscholes	100%	100%
bodytrack	92%	100%
ferret	100%	100%
fluidanimate	6%	31%
freqmine	100%	100%
swaptions	100%	100%
vips	100%	100%
x264	27%	100%

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

Outline

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
`https://github.com/eschulte/goa/tree/asplos2014`

Eric Schulte

email `eschulte@cs.unm.edu`

homepage `https://cs.unm.edu/~eschulte`

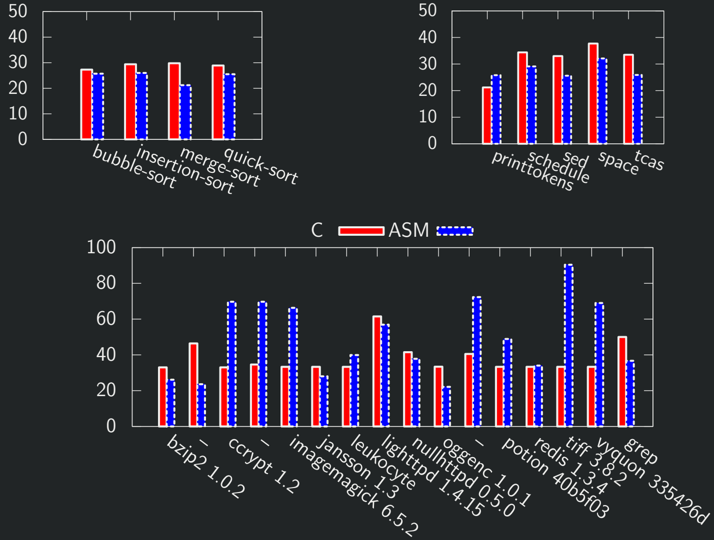
Backup Slides

Genetic Algorithm

Parameters

Parameter	Genprog	GOA
population size	40	2^{10}
evaluations	400	2^{18}
selection	fitness proportionate	tournament of 2
runtime	minutes	hours

Software Mutational Robustness



Program Syntactic Space