The Evolution of Extant Software

Eric Schulte

Department of Computer Science
University of New Mexico
Albuquerque, NM

2014

The Evolution of Extant Software

The Evolution of Extant Software

Extant Software

The existing Software ecosystem.

- applications
- libraries
- compilers
- operating systems
- architectures

The Evolution of Extant Software

Extant Software

The existing Software ecosystem.

- applications
- libraries
- compilers
- operating systems
- architectures

Evolution

Evolved product of evolutionary forces

Evolvable amenable to automated improvement

Genprog

Automatically Fix Bugs in C Software

Genprog

Automatically Fix Bugs in C Software

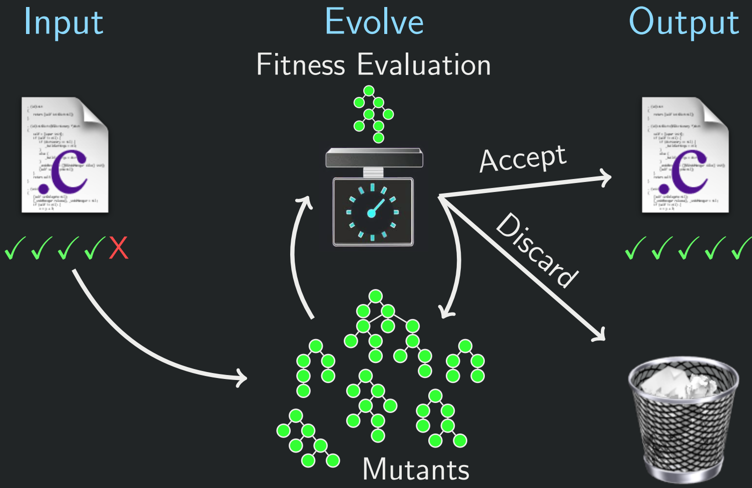
Collaboration between UNM and UVA

Dr. Stephanie Forrest

Dr. Westley Weimer

Genprog

Automatically Fix Bugs in C Software



Genprog

Automatically Fix Bugs in C Software

Strengths

Effective	Repaired 55/105 bugs for $8 each
General	Multiple classes of bugs and security defects
Best Papers	ICSE 2009, GECCO 2009, SBST 2009
Humies	Gold 2009, Bronze 2012

Genprog

Automatically Fix Bugs in C Software

But

Why doesn't this break my software?

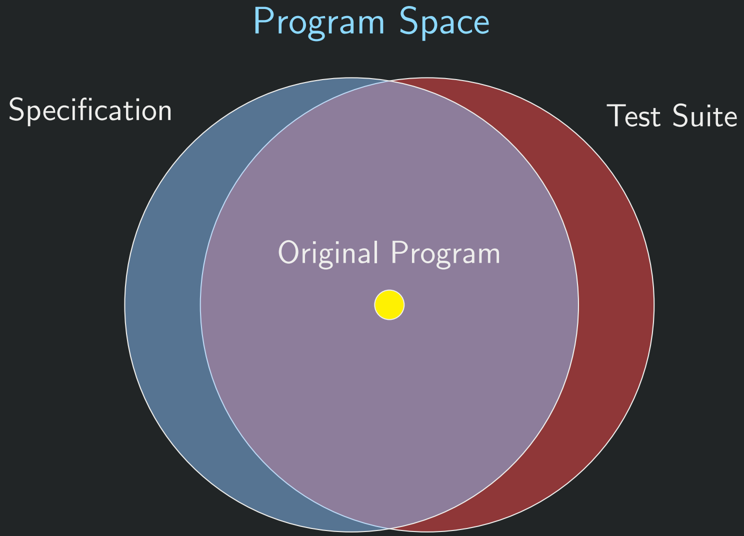
Outline

Software Mutational Robustness

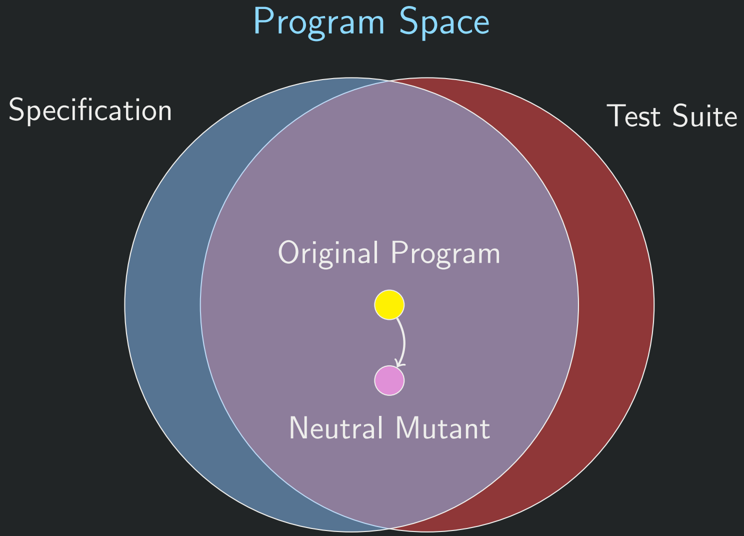
[Schulte, GPEM 2013]

percentage of mutants which are functional

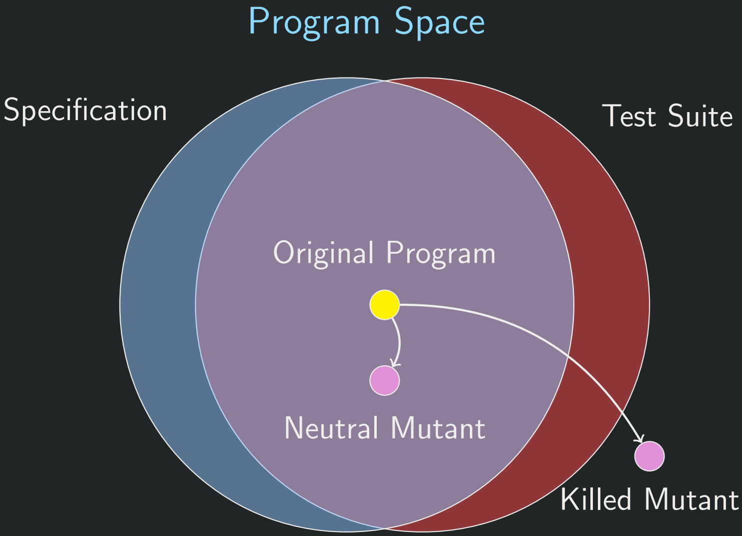
Software Mutational Robustness



Software Mutational Robustness



Software Mutational Robustness



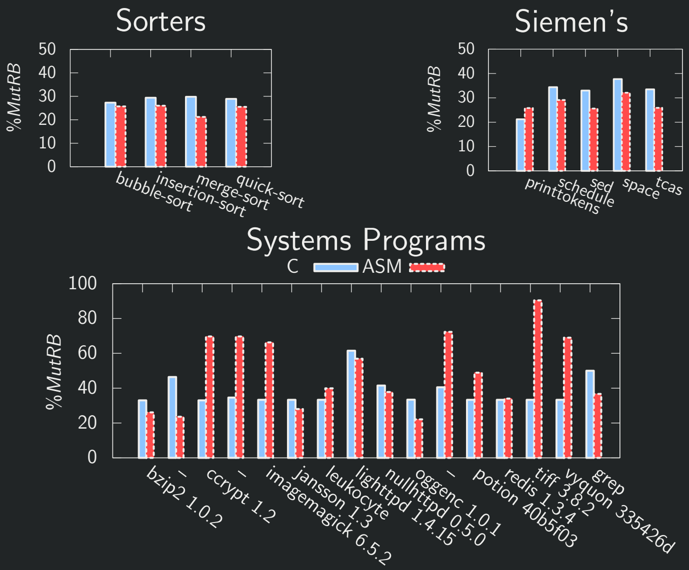
Definition

$MutRB(P, T, M)$

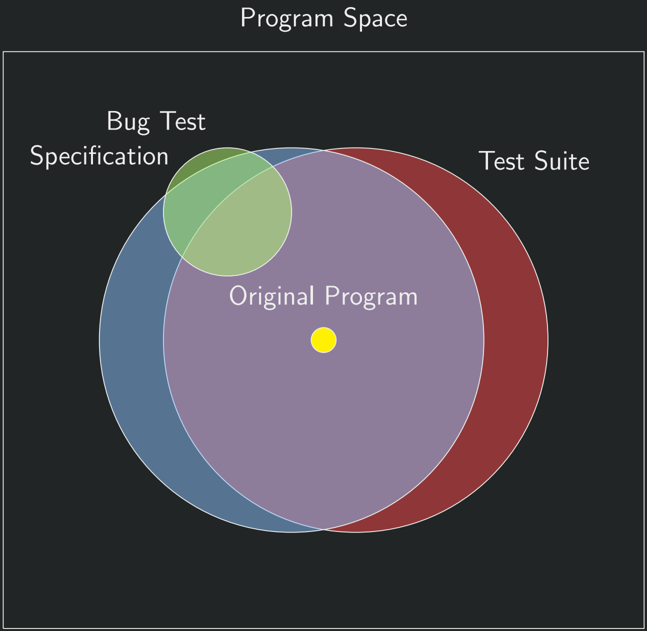
P	program
T	test suite
M	mutation operators

$$MutRB(P, T, M) = \frac{|\{P' \mid m \in M.\ P' \leftarrow m(P)\ \wedge\ T(P')\}|}{|\{P' \mid m \in M.\ P' \leftarrow m(P)\}|}$$

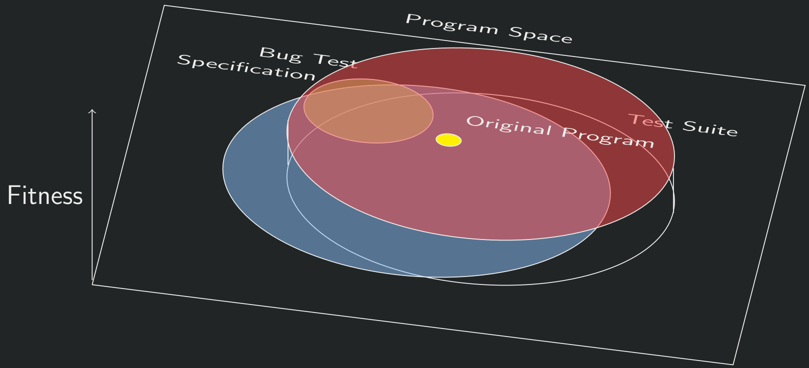
Measurements



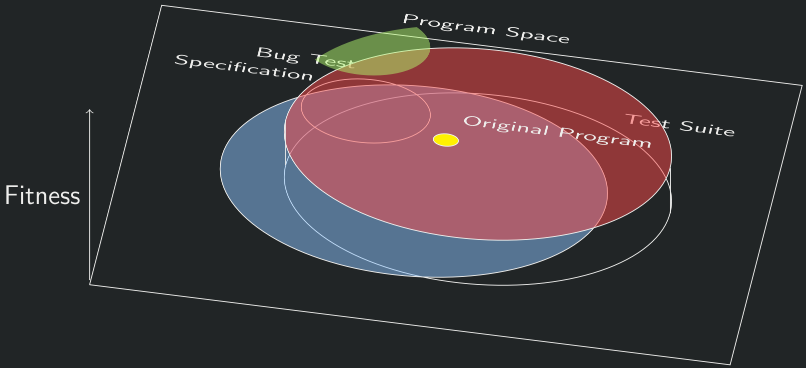
Program Repair



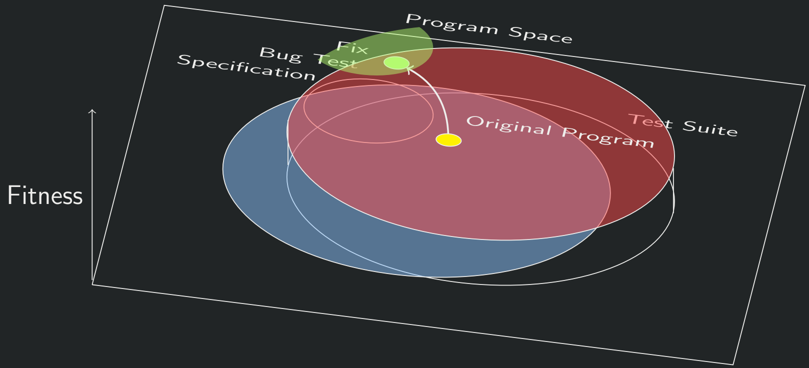
Program Repair



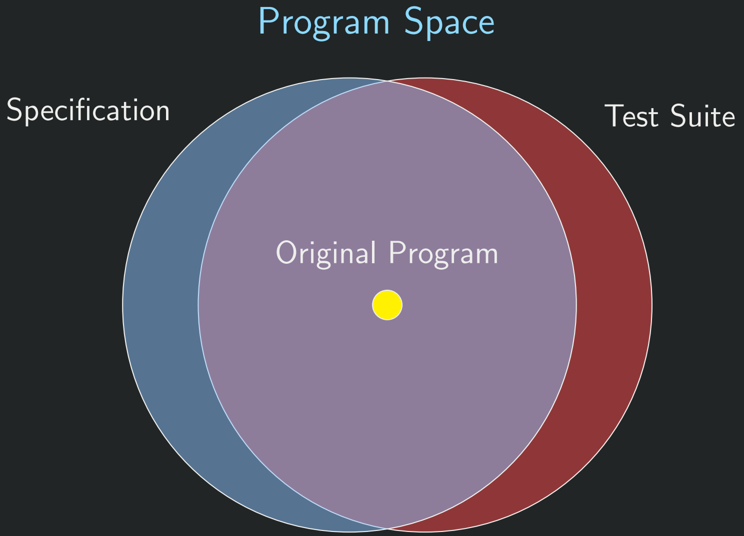
Program Repair



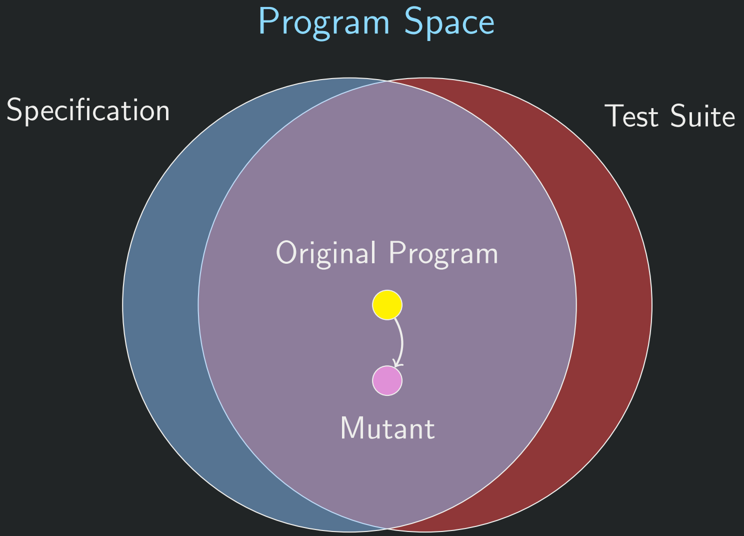
Program Repair



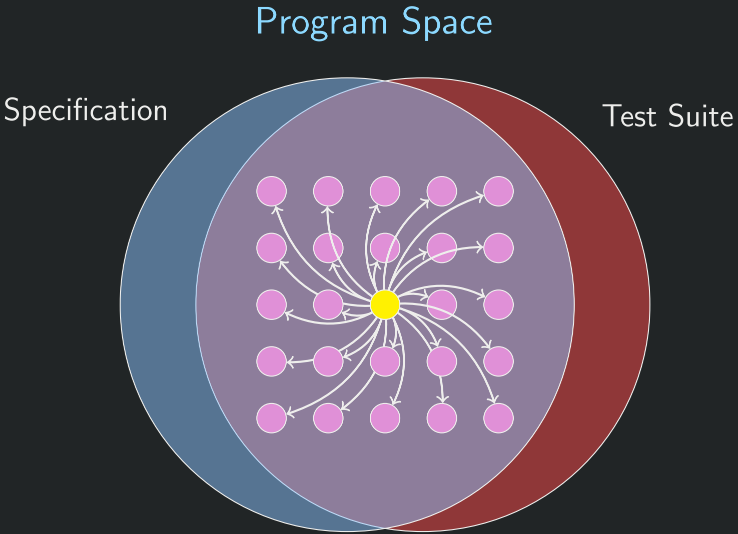
Automated Diversity



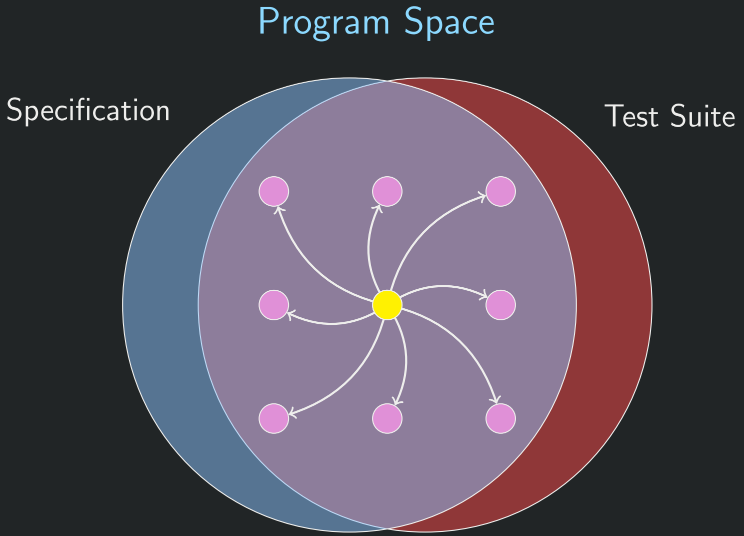
Automated Diversity



Automated Diversity

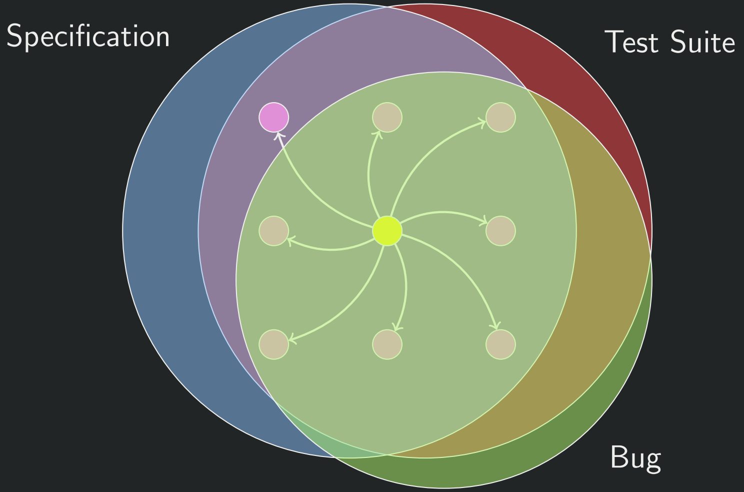


Automated Diversity



Automated Diversity and Proactive Bug Repair

Program Space



Outline

Program Representation

Source

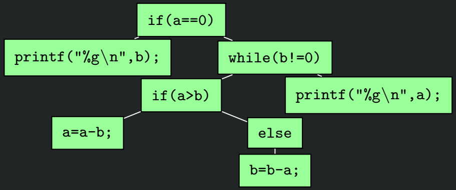
```
if (a==0){
  printf("%g\n", b); }
else {
  while (b!=0){
    if (a>b){ a=a-b; }
    else    { b=b-a; } } }
printf("%g\n", a);
```

Program Representation

Source

```
if (a==0){
  printf("%g\n", b); }
else {
  while (b!=0){
    if (a>b){ a=a-b; }
    else    { b=b-a; } } }
printf("%g\n", a);
```

AST

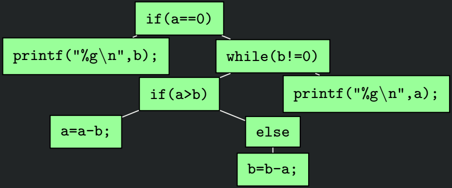


Program Representation

Source

```
if (a==0){
  printf("%g\n", b); }
else {
  while (b!=0){
    if (a>b){ a=a-b; }
    else    { b=b-a; } } }
printf("%g\n", a);
```

AST



ASM [Schulte, ASE 2010]

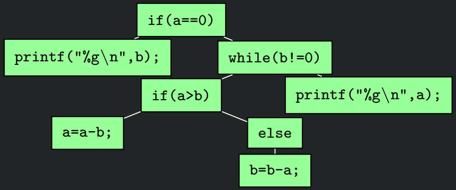
.file "gcd.c"
.globl main
.type main, @function
main:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $48, %rsp

Program Representation

Source

```
if (a==0){
  printf("%g\n", b); }
else {
  while (b!=0){
    if (a>b){ a=a-b; }
    else    { b=b-a; } } }
printf("%g\n", a);
```

AST



ASM [Schulte, ASE 2010]

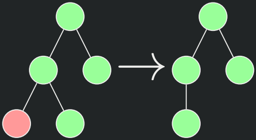
```
.file "gcd.c"
.globl main
.type main, @function
main:
.cfi_startproc
pushq %rbp
.cfi_def_cfa_offset 16
.cfi_offset 6, -16
movq %rsp, %rbp
.cfi_def_cfa_register 6
subq $48, %rsp
```

ELF [Schulte, ASPLOS 2013]

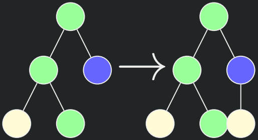
```
ELF\?
ELF header
program header table
section 1
...
.text section
[55] [48 89 e5] [48 83 ec 20] [48 89 7d e8] [89 75
e4] [83 7d e4 01] [7e 60] ...
...
section n
section header table
```

Program Mutation Operations

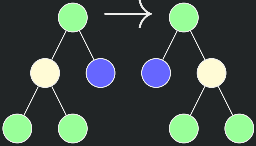
AST-Delete



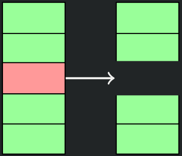
AST-Insert



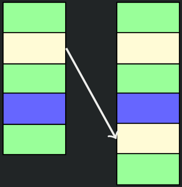
AST-Swap



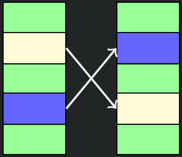
ASM-Delete



ASM-Insert



ASM-Swap



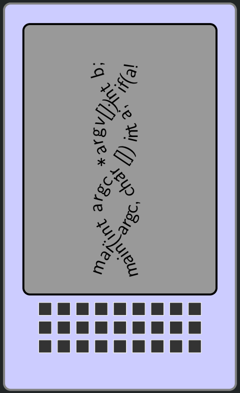
Outline

Program Repair in Embedded Devices

[Schulte, ASPLOS 2013]

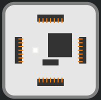
Resource Constraints

- Small disks
- Less memory
- Slow processors
- Slow, costly comm.



Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.



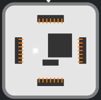
CPU

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movq %rdx, -80(%rbp)
addq $4, %r15
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
```

Machine-code Instructions

Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.



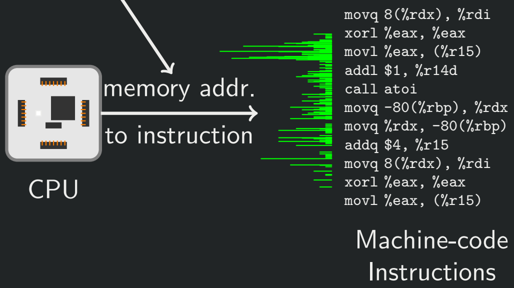
CPU

```
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
addl $1, %r14d
call atoi
movq -80(%rbp), %rdx
movq %rdx, -80(%rbp)
addq $4, %r15
movq 8(%rdx), %rdi
xorl %eax, %eax
movl %eax, (%r15)
```

Machine-code Instructions

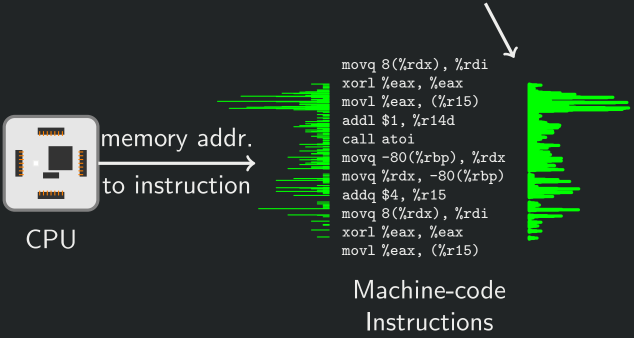
Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.



Light Weight Fault Localization

1. Sample program counter.
2. Translate memory addresses to program offsets.
3. Smooth sample with Gaussian convolution.



Genprog

Automatically Fix Bugs in C Software

Strengths

Effective	Repaired 55/105 bugs for $8 each
General	Multiple classes of bugs and security defects
Best Papers	ICSE 2009, GECCO 2009, SBST 2009
Humies	Gold 2009, Bronze 2012

Genprog

Automatically Fix Bugs in C Software

Strengths

Effective	Repaired 55/105 bugs for $8 each
General	Multiple classes of bugs and security defects
Best Papers	ICSE 2009, GECCO 2009, SBST 2009
Humies	Gold 2009, Bronze 2012

Limitations

- Requires source code
- Requires build tool chain
- Requires program instrumentation
- Expensive fitness function (compilation, test execution)

Genprog

Automatically Fix Bugs in C Software

Strengths

Effective	Repaired 55/105 bugs for $8 each
General	Multiple classes of bugs and security defects
Best Papers	ICSE 2009, GECCO 2009, SBST 2009
Humies	Gold 2009, Bronze 2012

Limitations

- ~~Requires source code~~
- ~~Requires build tool chain~~
- ~~Requires program instrumentation~~
- Expensive fitness function (~~compilation~~, test execution)

Embedded Repair Benchmark Programs

Program	Program Description	Bug
`atris`	graphical tetris game	stack buffer exploit
`ccrypt`	encryption utility	segfault
`deroff`	document processing	segfault
`flex`	lexical analyzer generator	segfault
`indent`	source code processing	infinite loop
`look svr4`	dictionary lookup	infinite loop
`look ultrix`	dictionary lookup	infinite loop
`merge`	merge sort	duplicate inputs
`merge-cpp`	merge sort (in C++)	duplicate inputs
`s3`	sendmail utility	buffer overflow
`uniq`	duplicate text processing	segfault
`units`	metric conversion	segfault
`zune`	embedded media player	infinite loop

Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Total bugs repaired

Rep.	Num. Bugs
AST	13
ASM	12
ELF	11

Average success rate

100 runs per bug

Rep.	Success Rate
AST	78.17%
ASM	70.75%
ELF	65.83%

Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Expected fitness evaluations

Rep.	Evaluations
AST	583.98
ASM	188.38
ELF	207.15

Total runtime

Rep.	Sec.
AST	229.50
ASM	278.30
ELF	74.20

Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Example: Merge Sort

Repair by representation

- AST, 2 of 4900 Swaps
- ASM, 1 of 280 Deletes

merge.c

```
if(left[l-mid-1]<=right[0]){
  result=list; }
else{/* fix:swap branches */
  result=merge(left,l-mid,
               right,mid); }
```

merge.s

```
cmpl %eax, %edx ; fix: del.
jg   .L12
movq -72(%rbp), %rax
```

Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Disk size

Rep.	Requirements
AST	Source code & build toolchain
ASM	Assembly code & linker
ELF	Compiled executable

Embedded Repair Results

- Effective
- 62% faster runtime
- 95% smaller disk footprint
- 86% less memory

Working memory

Rep.	MB
AST	1402
ASM	756
ELF	200

Outline

NETGEAR Exploit

[Schulte, unpublished]

1. URI starting with `BRS` bypasses authentication
2. URI including `unauth.cgi` or `securityquestions.cgi` bypass authentication
3. unprotected page removes authentication for every page `http://router/BRS_02_genieHelp.html`

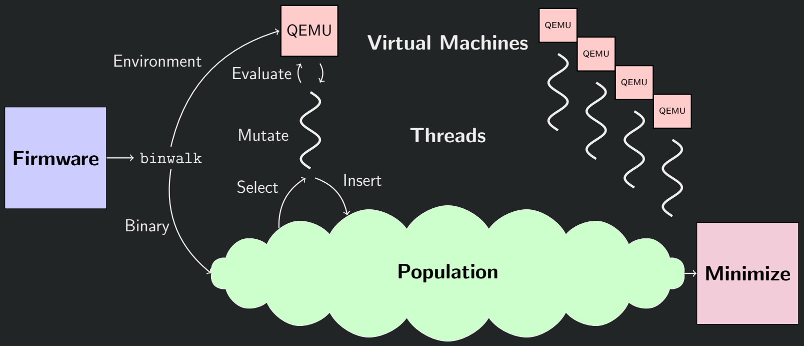
Common Vulnerability

NETGEAR WNDR4700

D-Link DIR-100

Linksys WAG200G

NETGEAR Repair Overview



NETGEAR Repair Results

Results

- repair without regression tests
- repair of real un-patched exploit
- multiple concurrent repairs

Repair Runtime

evaluations	$\sim 36,000$
runtime	86.6 minutes
threads	32

Outline

Optimization Problem

[Schulte, ASPLOS 2014]

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 (GOA)

- 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

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

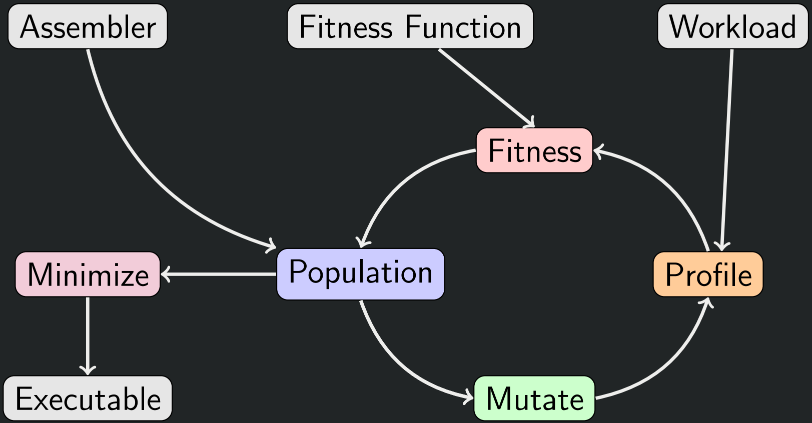
Use test cases to exercise program

- evaluate functionality
- measure runtime properties

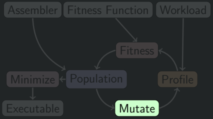
Technique

Post-compiler, test-driven, Genetic Optimization Algorithm

Genetic Optimization Algorithm (GOA)



Program Mutation



Software Representation

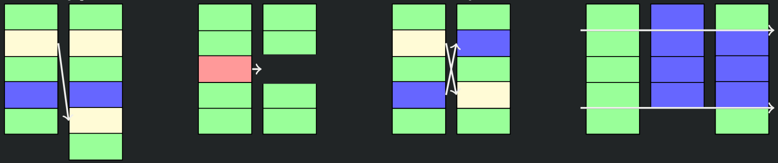
Raw Assembler Text

```
    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
```

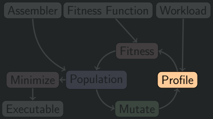
```
 movl  $0, -8(%rbp)
.L4:
 addl  $1, -4(%rbp)
.L3:
 movl  -4(%rbp), %eax
 cmpl  -28(%rbp), %eax
 jl   .L5
```

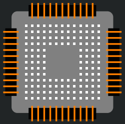
Mutation Operations

Copy Delete Swap Two Point Crossover



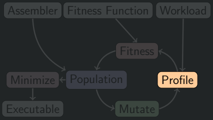
Profiling





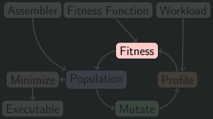
Hardware
Performance
Counters

Profiling



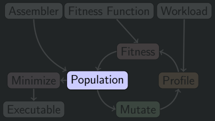

```
$ perf stat -- ./blackscholes 1 input /tmp/output

  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

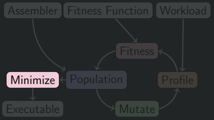


Details

- population size: 2^{10}
- 2^{18} fitness evaluations
- ~ 16 hour runtime per optimization

Minimization

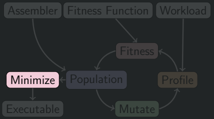
Delta Debugging




```
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

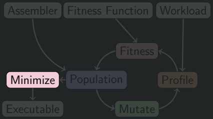
Delta Debugging




```
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

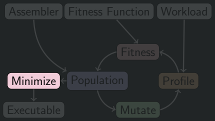
Delta Debugging




```
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

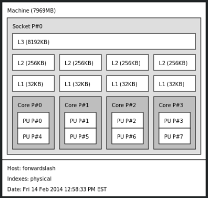
Program	C/C++ Lines of Code	ASM Lines of Code	Description
blackscholes	510	7,932	Finance modeling
bodytrack	14,513	955,888	Human video tracking
facesim			no alternate inputs
ferret	15,188	288,981	Image search engine
fluidanimate	11,424	44,681	Fluid dynamics animation
freqmine	2,710	104,722	Frequent itemset 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



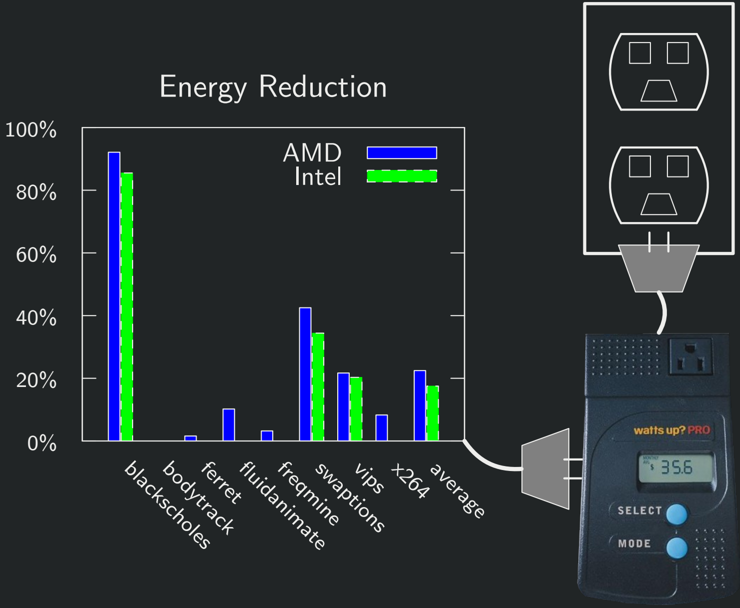
Energy Model

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

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



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

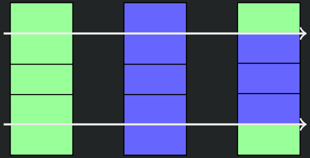
Future Work

Heterologous crossover, Verification, Static, Hardening

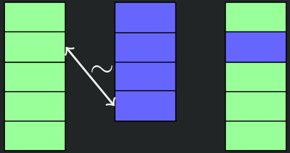
Crossover between

- different compilation flags
- different versions
- different implementations
- different programs

Homologous Crossover



Heterologous Crossover



Future Work

Heterologous crossover, Verification, Static, Hardening

- Assembler diff chunks formally equivalent
- Invariant preservation
  e.g., Daikon

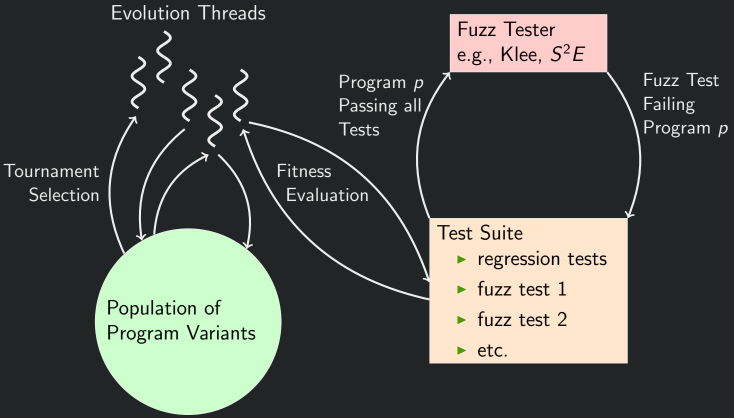
Future Work

Heterologous crossover, Verification, Static, Hardening

- Static analysis tools
- Annotations
  e.g., Eiffel contracts, Meta-level compilation

Future Work

Heterologous crossover, Verification, Static, **Hardening**



Outline

Conclusion

Extant software is

1. Mutationally robust

Conclusion

Extant software is

1. Mutationally robust
2. A product of evolution

Conclusion

Extant software is

1. Mutationally robust
2. A product of evolution
3. Amenable to automated evolutionary improvement

Thank You

Eric Schulte

email `eschulte@cs.unm.edu`

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

Backup Slides

Slides

- Benefits of ASM
- Embedded Repair Search Space
- Many Bugs
- LLVM Representation
- Optimization: Runtime and Energy Reduction

Benefits of ASM

Back

The ASM representation performs well.

Avoids

- direct addresses
- argumented instructions

Similarities to DNA

- Linear vector genome
- read sequentially
- reading frames
- start and stop codes
- padding
- bootstrapped compilers

Embedded Repair Search Space

Back

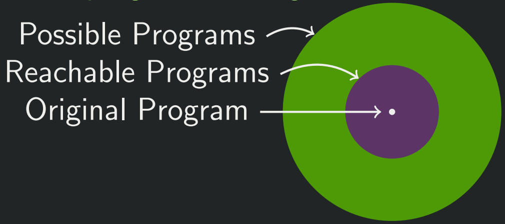
Program size

$\approx 3\times$ more assembly instructions than C statements

Search space size

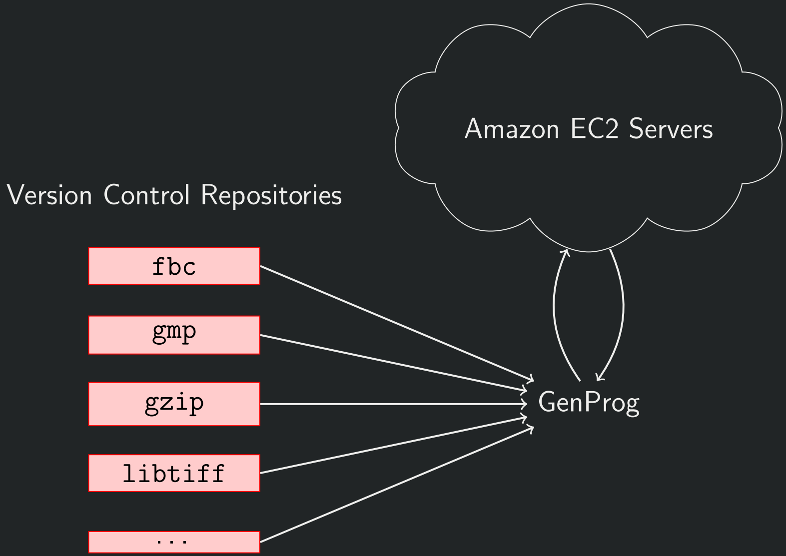
$= |\text{alphabet}|^{\text{program size}}$

Possible program coverage



Systematic Evaluation

[Le Goues, ICSE 12] Back



Fixing Bugs for $8 a Bug

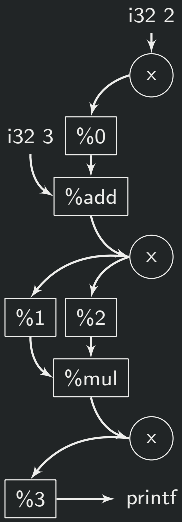
Back

Program	Defects Repaired	Avg. Cost per Non-Repair Hours	Avg. Cost per Non-Repair US$	Avg. Cost Per Repair Hours	Avg. Cost Per Repair US$
`fbc`	1 / 3	8.52	5.56	6.52	4.08
`gmp`	1 / 2	9.93	6.61	1.60	0.44
`gzip`	1 / 5	5.11	3.04	1.41	0.30
`libtiff`	17 / 24	7.81	5.04	1.05	0.04
`lighttpd`	5 / 9	10.79	7.25	1.34	0.25
`php`	28 / 44	13.00	8.80	1.84	0.62
`python`	1 / 11	13.00	8.80	1.22	0.16
`wireshark`	1 / 7	13.00	8.80	1.23	0.17
total	**55 / 105**	**11.22h**		**1.60h**	

LLVM Representations

Back

```
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
```

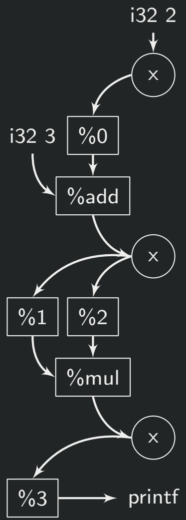


LLVM Delete

Back

delete 4

```
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %0, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
```

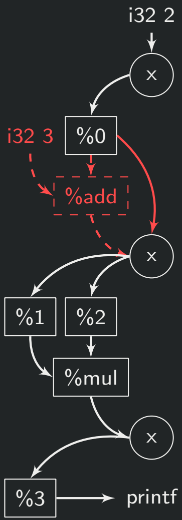


LLVM Delete

Back

delete 4

```
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %0, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
```

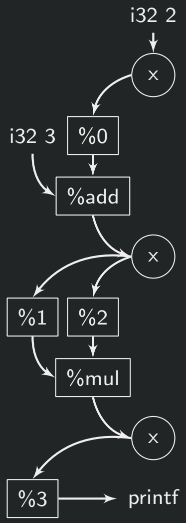


LLVM Insert

Back

insert 9 4

```
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2

store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
```

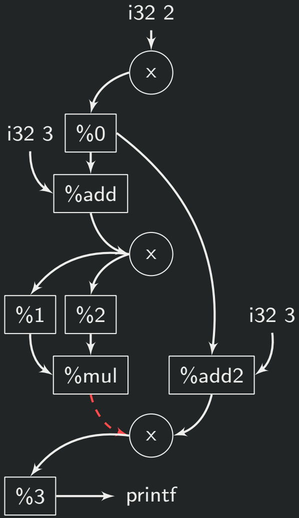


LLVM Insert

Back

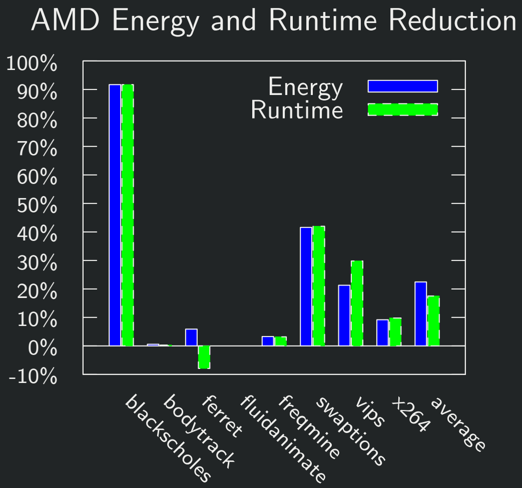
insert 9 4

```
%x = alloca i32, align 4
store i32 2, i32* %x, align 4
%0 = load i32* %x, align 4
%add = add nsw i32 %0, 3
store i32 %add, i32* %x, align 4
%1 = load i32* %x, align 4
%2 = load i32* %x, align 4
%mul = mul nsw i32 %1, %2
%add.insert = add nsw i32 %0, 3
store i32 %mul, i32* %x, align 4
%3 = load i32* %x, align 4
%call = call ... @printf(@.str %3)
```



Results: Runtime and Energy Reduction

Back



The End

Back

Eric Schulte

email `eschulte@cs.unm.edu`

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