Superoptimization

Can compiler optimization be seen as a search problem?

Raghuveer S (Raghu)

MSCS, CU Boulder

A bit about me

- MS in Computer Science
- Software Engineer with Hewlett-Packard (Distributed Systems)
- Software Engineer with Synopsys Inc (VLSI)
- **Interest:** Compilers $\cap$ Distributed Systems

Outline

This Presentation

- Intro to Optimization (20 min).
- Optimization and the search space (5 min).
- What is Superoptimization and Why do we care? (10 min).
- Stoke and Souper (10 min)

Intro to Optimization

Optimization problem. Given a set X of candidate solutions, and a criterion map $F : X \to R$, find an $x \in X$ such that $x \in argmax_{y \in X} F(y)$

Eg,

- **Domain:** Set of `stack` locations assigned to variables
- **Codomain:** Non-interfering sets of `registers` that can be mapped to the `stack` locations.

Intro to Optimization Contd..

`F`: Function that maps most number of registers to stack locations.

- Register Allocation
  - Efficient Coloring, Dominance, LVN Based Liveness, DSE, etc

Intro to Optimization

Stack vs Register

```
x = 2
print 2
```

Stack

```
mov $2, -10(%ebp)
pushl -10(%ebp)
call print
```

Register

```
mov $2, %eax
pushl %eax
call print
```

Intro to Optimization

Coloring

- Every register gets a color
- Every stack location gets a color

Algorithm

- Construct a graph
- Insert all the variables as nodes in the graph
- Assign a color to each variable
- If there are more variables than registers, spill them over to stack.

Intro to Optimization

Efficient Coloring

- Handle spilling by reassigning the stack location to a register.

```
mov $2, -4(%ebp)
mov -4(%ebp), %eax
pushl %eax
call print
```

Intro to Optimization

Coloring with interference

- Analyze liveness (on IR)

$$live_{before}(inst) = (live_{after}(inst) - write(inst)) \cup read(inst)$$

```
mov $2, x
mov $4, y
addl $7, x
print x
```

Intro to Optimization

Coloring with interference

- Analyze liveness (on IR)

$$live_{before}(inst) = (live_{after}(inst) - write(inst)) \cup read(inst)$$

```
{}
mov $2, x
{x}
mov $4, y
{x, y}
addl y, x
{x}
print x
{}
```

We know that x and y interfere!

Intro to Optimization

VN Based Liveness

```
x = 2
y = 3
z = x + y
w = x + y
```

```
mov 2, x
mov 3, y
mov x, z
add y, z
mov x, w
add y, w
```

Intro to Optimization

VN Based Liveness

```
x = 2
y = 3
z = x + y
w = x + y
```

```
{}
mov 2, x
{x}
mov 3, y
{x, y}
mov x, z
{x, y, z}
add y, z
{x, y}
mov x, w
{y, w}
add y, w
{}
```

Intro to Optimization

VN Based Liveness

```
x (1) = 2 (1)
y (2) = 3 (2)
z (3) = x (1) + y (2)
w (?) = x (3) + y (2)
---> Look in the Hash Table
```

```
{}
mov 2 (1), x (1)
{x}
mov 3 (2), y (2)
{x, y}
mov x (1), z (1)
{x, y, z}
add y (2), z (3)
{x, y}
mov x (1), w (1)
{y, w}
add y (2), w (?)
-----> Look in the Hash Table
{}
```

Benefit: w does not have considered for register allocation.

Intro to Optimization

VN Based Liveness and DSE

```
x = 2
y = 3
z = x + y
w = x + y
```

```
{}
mov 2, x
{x}
mov 3, y
{x, y}
mov x, z
{x, y, z}
add y, z
{x, y}
mov x, w
{y, w}
add y, w
{}
```

Intro to Optimization
VN Based Liveness and DSE

```
x = 2
y = 3
z = x + y
w = x + y
```

```
{}
mov 2, x
{x}
mov 3, y
{x, y}
mov x, z
{x, y, z}
add y, z
{x}
mov x, w
{}
```

Intro to Optimization

VN Based Liveness and DSE

```
x = 2
y = 3
z = x + y
w = x + y
```

```
{}
mov 2, x
{x}
mov 3, y
{x, y}
mov x, z
{y, z}
add y, z
{}
```

Intro to Optimization

VN Based Liveness and DSE

```
x = 2
y = 3
z = x + y
w = x + y
```

```
{}
mov 2, x
{x}
mov 3, y
{x}
mov x, z
{}
```

Intro to Optimization

VN Based Liveness and DSE

```
x = 2
y = 3
z = x + y
w = x + y
```

```
{}
mov 2, x
{}
mov 3, y
{}
```

Intro to Optimization

VN Based Liveness and DSE

```
x = 2
y = 3
z = x + y
w = x + y
```

{}

Intro to Optimization

VN Based Liveness and DSE

```
x = 2
y = 3
z = x + y
w = x + y
print w
print z
```

This would completely change our DSE

Control Flow

`If Statements, Loops, etc`

- Basic Blocks
- CFG
- LVN
- Dominance

Control Flow

CFG and Basic Blocks

```
if input():
  foo()
else:
  baz()
```

- call input
- cmp %eax, $1
- jne else
- then:
- call foo
- jmp endif
- else:
- call baz
- jmp endif

Control Flow

CFG and Basic Blocks

```
if input():
  foo()
else:
  baz()
```

Block 1:

- call input
- cmp %eax, $1
- jne else

Block 2:

- then:
- call foo
- jmp endif

Block 3:

- else:
- call baz
- jmp endif

Control Flow

CFG and Basic Blocks

```
x = 2
y = 0
while x != 12:
  x = y + 1
  y = x + 1
print x
```

- mov 2, x
- mov 0, y

- while:
- cmp x, 12
- je endwhile

- mov x, y
- add 1, x
- mov x, y
- add 1, y
- jmp while

- endwhile:
- print x

What do we do?

- Advanced optimizations cannot be done in the above code because ?
- Solution: SSA + Dominance + Phi - Explain Store, Definition, and SSA (Dominance is out of scope)

```
x = 2
y = 0
while x != 12:
  x = y + 1
  y = x + 1
print x
```

```
x1 = 2
y1 = 0
while x(?) != 12:
  x2 = y(?) + 1
  y2 = x2 + 1
print x(?)
```

What do we do?

```
x = 2
y = 0
while x != 12:
  x = y + 1
  y = x + 1
print x
```

```
x1 = 2
y1 = 0
while x(phi(x1, x2)) != 12:
  x2 = y(phi(y1, y2)) + 1
  y2 = x2 + 1
print x(phi(x1, x2))
```

Optimization and the search space

Thinking of optimization as a search problem?

- Pure search problems only need verification of the solution.
- Optimization problems need verification of the entire domain.

Superoptimization

- Optimization problem as search problem

Superoptimization

- What is the search space here?
  - The computational universe - space of all possible programs

Superoptimization

- Perturbation
  - mul $\rightarrow$ shifts $+$ movs $+$ adds (depends upon context)
  - control flow $\rightarrow$ add $+$ carry

Superoptimization

- Correctness
  - Equivalence?

Superoptimization

- Performance
  - Speed up?
  - Less Memory?

Superoptimization

- Applicability
  - ▪ X - (X - Y) $\rightarrow$ Y
  - ▪ (X«1) - X $\rightarrow$ X

Superoptimization

- Limitations
  - Size - Verification, Compile Time
  - Complexity - Control Flow

Stoke

Stochastic Superoptimizer

- Only works on Loop Free Code.
- Treats program optimization task as a cost minimization problem.

Stoke

Stochastic Superoptimizer

- Verification?
  - C(R; T): Eq(R; T) + Perf(R; T)

Stoke

Stochastic Superoptimizer

- Eq(.)
  - Z3: SAT Solver

Stoke

Stochastic Superoptimizer

- Perf(.)
  - MCMC: Metropolis Hasting

Stoke

Stochastic Superoptimizer

```
# gcc -O3

.L0:
  movq rsi, r9
  movl ecx, ecx
  shrq 32, rsi
  andl 0xffffffff, r9d
  movq rcx, rax
  movl edx, edx
  imulq r9, rax
  imulq rdx, r9
  imulq rsi, rdx
  imulq rsi, rcx
  addq rdx, rax
  jae .L2
  movabsq 0x100000000, rdx
  addq rdx, rcx
.L2:
  movq rax, rsi
  movq rax, rdx
  shrq 32, rsi
  salq 32, rdx
  addq rsi, rcx
  addq r9, rdx
  adcq 0, rcx
  addq r8, rdx
  adcq 0, rcx
  addq rdi, rdx
  adcq 0, rcx
```

```
# STOKE

.L0:
  shlq 32, rcx
  movl edx, edx
  xorq rdx, rcx
  movq rcx, rax
  mulq rsi
  addq r8, rdi
  adcq 0, rdx
  addq rdi, rax
  adcq 0, rdx
  movq rdx, r8
  movq rax, rdi
```

Souper

First attempt at using superoptimizers in code generation.

- LLVM IR
- Phi Node exploitation
- Redis + Temp KV Store

Souper

- Straight line optimization (Deterministic) vs Novel Logic(Superoptimization)
- Path Dependence vs Path Linearity

Souper

Example Result

```
%a:i64 = var
%x:i64 = var
%y:i64 = var
%i = eq %a, %x
%j = ne %a, %y
%r = and %i, %j
infer %r
```

(a)

```
%z = slt %x, %y
pc %z 1
```

(b)

```
result %i
```

(c)

Future Interest

- Transformers and Superoptimization Eg. Codex by OpenAI (Github Copilot)

QA