A Quest Toward the Perfect Optimizing Compiler

Theodoros Theodoridis
Compiler
BOLT: A Practical Binary Optimizer for Data Centers and Beyond

Maksim Panchenko, Rafael Auber, Bill Nell, Guillaume Ossosi
Pactware, Inc.
Menlo Park, CA, USA

How far are we from the optimum? Pretty far…

1.3x Speedup

Up to 5x Speedup

mold: A Modern Linker

mold is a faster drop-in replacement for existing Unix linkers. It is several times faster than the LLVM ld.lnk, the second-fastest open-source linker which I originally created a few years ago. mold is designed to increase developer productivity by reducing build time, especially in rapid debug-edit-rebuild cycles.

Here is a performance comparison of GNU gold, LLVM ld.lnk, and mold for linking final debuginfo-enabled executables of major large programs on a simulated 8-core 16-threads machine.

<table>
<thead>
<tr>
<th>Time per build (shorter is better)</th>
</tr>
</thead>
<tbody>
<tr>
<td>GNU gold</td>
</tr>
<tr>
<td>Chrome 96 (1.89 GB)</td>
</tr>
<tr>
<td>Clang 13 (3.18 GB)</td>
</tr>
<tr>
<td>Firefox 89 (1.64 GB)</td>
</tr>
</tbody>
</table>

mold is so fast that it is only 2x slower than on the same machine. Feel free to file a bug if you find mold is not faster than other linkers.
Our Approach

1. We obtain the optimum.

2. We compare with the compiler and find the gap.

“What if we had optimal ...?”
"What if we had optimal inlining?

Finding Missed Optimizations through the Lens of Dead Code Elimination

Through the Lens of Dead Code Elimination

Heuristics

Transformations

Analyses

DCE CSE INL DA VR

AA

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

Heuristics

Understanding and Exploiting Optimal Function Inlining

What if we had optimal inlining?

Finding Missed Optimizations through the Lens of Dead Code Elimination

Through the Lens of Dead Code Elimination

Heuristics

Transformations

Analyses

DCE CSE INL DA VR

AA

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

Heuristics

Understanding and Exploiting Optimal Function Inlining

What if we had optimal inlining?

Finding Missed Optimizations through the Lens of Dead Code Elimination

Through the Lens of Dead Code Elimination

Heuristics

Transformations

Analyses

DCE CSE INL DA VR

AA

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

Heuristics

Understanding and Exploiting Optimal Function Inlining

What if we had optimal inlining?

Finding Missed Optimizations through the Lens of Dead Code Elimination

Through the Lens of Dead Code Elimination

Heuristics

Transformations

Analyses

DCE CSE INL DA VR

AA

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

Heuristics

Understanding and Exploiting Optimal Function Inlining

What if we had optimal inlining?

Finding Missed Optimizations through the Lens of Dead Code Elimination

Through the Lens of Dead Code Elimination

Heuristics

Transformations

Analyses

DCE CSE INL DA VR

AA

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ... 

... ... ... ...
The Benefits of Inlining

```c
int bar(int a, int b) {
    if ((a * b) % 2)
        return a + b;
    else
        return a - b;
}

int foo(int x) {
    return bar(x, 2) + 2;
}
```

After inlining:

```c
int foo(int x) {
    return x;
}
```
Too much Inlining is Bad

Aggressive Inlining: 69% binary size increase
Proper Inlining Reduces Program Size

Relative size: clang -Os vs clang -Os -fno-inline

Up to 3x improvement

SPEC CPU 2017 benchmarks
Gap between LLVM and Optimal

Heuristic size overhead

- Max 281%
- 8.5% of cases with 10+% overhead
- LLVM finds the optimal in 526 cases (not shown)

Call graphs extracted from SPEC 2017 (n=1,135)

Common inlining choices

- None inline
- Only LLVM inlines
- Only optimal inlines
- Both inline

LLVM’s heuristic is too aggressive
“What if we had optimal DCE?”

Transformations

- DCE
- CSE
- INL

Analyses

- AA
- DA
- VR

Theodore Theodorides
theodore.theodorides@inf.eth.ch
ETH Zurich
Switzerland

Manuel Rigger
manuel.rigger@inf.eth.ch
ETH Zurich
Switzerland

Zhendong Su
zhendong.su@inf.eth.ch
ETH Zurich
Switzerland

ABSTRACT

Finding missed optimizations through the Lens of Dead Code Elimination. We present a novel approach to discover and automatically identify opportunities for improving compiled code. Our method is based on analyzing the effects of dead code elimination on the performance of real-world applications. The approach is intended to provide practical insights into how dead code elimination can be used to improve the performance of compiled code. Our method is based on analyzing the effects of dead code elimination on the performance of real-world applications. The approach is intended to provide practical insights into how dead code elimination can be used to improve the performance of compiled code.
static int a = 0;

int main () {
    if (a != 0) {
        return 1;
    }
    a = 1;
    return 0;
}

main:
xorl %eax, %eax
retq
```c
static int a = 0;

int main () {
    if (a != 0) {
        return 1;
    }
    a = 1;
    return 0;
}
```
Dead Code Elimination: An Optimization Sink

- CSE
- Const. Prop.
- Alias Analysis
- ...
How good are compilers at DCE?

Corpus of 10,000 test programs:

- Generated with Csmith
- 3,109,167 dead blocks

<table>
<thead>
<tr>
<th>Optimization Level</th>
<th>% of dead blocks that are missed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>GCC</td>
</tr>
<tr>
<td>O0</td>
<td>85.2%</td>
</tr>
<tr>
<td>O1</td>
<td>8.2%</td>
</tr>
<tr>
<td>Os</td>
<td>6.0%</td>
</tr>
<tr>
<td>O2</td>
<td>5.7%</td>
</tr>
<tr>
<td>O3</td>
<td>5.6%</td>
</tr>
</tbody>
</table>
Finding Missed Optimization Opportunities Automatically

<table>
<thead>
<tr>
<th></th>
<th>LLVM</th>
<th>GCC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reported</td>
<td>47</td>
<td>55</td>
</tr>
<tr>
<td>Confirmed</td>
<td>35</td>
<td>46</td>
</tr>
<tr>
<td>Fixed</td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>
Our Approach

1. We obtain the optimum.

2. We compare with the compiler and find the gap.

“What if we had optimal ...?”

Ongoing Work

- Optimal Alias Analysis Information
- Optimal Pass Pipelines
- Learning Heuristics based on Optimal Inlining Choices
Backup Slides
This can be done recursively!

Total: \((2^2 + 2^2 + 1) + 2^4 = 25 < 32\) naïve
Lens of Dead Code Elimination
int a = 0;
static int b[2] = {0,0}, c = 0;

```c
int main() {
    if (b[a]) {
        return 1;
    }
    if (c) {
        return 2;
    }
    c = 1;
    return 0;
}
```
Missed Dead Code Elimination Detection

```
int a = 0;
static int b[2] = {0,0}, c = 0;

int main() {
    if (b[a]) {
        return 1;
    }
    if (c) {
        return 2;
    }
    c = 1;
    return 0;
}
```
int a = 0;
static int b[2] = {0,0}, c = 0;
int main() {
    if (b[a]) {
        DCEMarker1();
        return 1;
    }
    if (c) {
        DCEMarker2();
        return 2;
    }
    c = 1;
    return 0;
}
The Lens of Dead Code Elimination

Alive Markers:
- Marker1
- Marker2

Missed Optimization

Alive Markers:
- Marker1

ASM 1
- Marker1
- Marker2

ASM 2
- Marker1
DCE Examples
static int a[2], b, *c[2];
int main() {
    for (b = 0; b < 2; b++) {
        c[b] = &a[1];
    }
    if (!c[0]){
        DCEMarker();
    }
    return 0;
}
static long a = 78240;
static int b, d;
static short e;
static short c(short f, short h) {
    return h == 0 ||
    (f && h == 1) ? 0 : f % h;
}
int main() {
    short g = a;
    for (b = 0; b < 1; b++) {
        e = a;
        d = c((e == a) ^ g, a);
    }
    if (d) {
        DCEMarker();
        for (; a; a++);
    }
}
static int b = -1, e = 1;
static short c = 0, d = 0;
short a(unsigned short f, int g) {
    return f >> g;
}

int main() {
    c++;
    d = a(4294967295 + (c > 0), 1);
    e ^= (short)(d * 3) / (unsigned)b;
    if (!e)
        DCEMarker();
    e != 0
}
[SimplifyCFG] don't sink common insts too soon (PR34603)

This should solve:
https://bugs.llvm.org/show_bug.cgi?id=34603
...by preventing SimplifyCFG from altering redundant instructions before early-cse has a chance to run.