Optimal and Heuristic Global Code Motion for Minimal Spilling

Gergö Barany
gergo@complang.tuwien.ac.at

Institute of Computer Languages
Vienna University of Technology

POPL 2013 Student Session
January 23, 2013
(paper at CC 2013, Rome, March 2013)
Solve global code motion and register allocation as an integrated problem.
Global code motion

\begin{align*}
\text{start:} & \\
& i0 := 0 \\
& a := \text{read()} \\
\text{loop:} & \\
& i1 := \phi(i0, i2) \\
& b := a + 1 \\
& i2 := i1 + b \\
& c := f(a) \\
& \text{compare } i2 < c \\
& d := i2 \times 2 \\
& \text{blt loop} \\
\text{end:} & \\
& \text{return } d
\end{align*}
Global code motion

start:
   i0 := 0
   a := read()
loop:
   i1 := \phi(i0, i2)
   b := a + 1    \text{ loop invariant }
   i2 := i1 + b
   c := f(a)
   compare i2 < c
   d := i2 \times 2
   blt loop
end:
   return d
start:
    i0 := 0
    a := read()
loop:
    i1 := $\phi(i0, i2)$
    b := a + 1  \hspace{1cm} \text{loop invariant}
    i2 := i1 + b
    c := f(a)
    compare i2 < c
    d := i2 \times 2  \hspace{1cm} \text{partially dead}
    blt loop
end:
    return d
Global code motion

start:
  i0 := 0
  a := read()
loop:
  i1 := \phi(i0, i2)
  b := a + 1
  i2 := i1 + b
  c := f(a)
  compare i2 < c
  d := i2 \times 2
  blt loop
end:
  return d

start:
  i0 := 0
  a := read()
loop:
  b := a + 1
  i1 := \phi(i0, i2)
  i2 := i1 + b
  c := f(a)
  compare i2 < c
  d := i2 \times 2
  blt loop
end:
  return d
Global code motion

```
start:
i0 := 0
a := read()
loop:
i1 := \phi(i0, i2)
b := a + 1
i2 := i1 + b
b := a + 1
i2 := i1 + b
c := f(a)
c := f(a)
compare i2 < c
compare i2 < c
d := i2 \times 2
d := i2 \times 2
blt loop
blt loop
end:
end:
return d
return d
```
Register allocation: conflict graphs

original program

- a
- b
- c
- i
- d
Register allocation: conflict graphs

original program

allocation to 3 registers possible

Gergö Barany (TU Vienna) | Global Code Motion for Minimal Spilling
Register allocation: conflict graphs

original program

allocation to 3 registers possible

after global code motion

not 3-colorable!
Compute code motions and overlaps

```plaintext
start:
0: i0 := 0
1: a := read()
loop:
2: i1 := φ(i0, i2)
3: b := a + 1
4: i2 := i1 + b
5: c := f(a)
6: compare i2 < c
7: d := i2 × 2
8: blt loop
end:
9: return d
```

Avoidable overlaps

<table>
<thead>
<tr>
<th>Pair</th>
<th>Overlapping placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, d</td>
<td>7 in loop</td>
</tr>
<tr>
<td>b, c</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, d</td>
<td>3 in start, 7 in loop</td>
</tr>
<tr>
<td>b, i0</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, i2</td>
<td>3 in start</td>
</tr>
<tr>
<td>c, d</td>
<td>7 in loop, 7 before 6</td>
</tr>
<tr>
<td>d, i2</td>
<td>7 in loop</td>
</tr>
</tbody>
</table>
Compute code motions and overlaps

start:
0: \(i_0 := 0\)
1: \(a := \text{read()}\)

loop:
2: \(i_1 := \phi(i_0, i_2)\)
3: \(b := a + 1\)
4: \(i_2 := i_1 + b\)
5: \(c := f(a)\)
6: compare \(i_2 < c\)
7: \(d := i_2 \times 2\)
8: \(\text{blt loop}\)

end:
9: return \(d\)

Avoidable overlaps

<table>
<thead>
<tr>
<th>Pair</th>
<th>Overlapping placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a, d)</td>
<td>7 in loop</td>
</tr>
<tr>
<td>(b, c)</td>
<td>3 in start</td>
</tr>
<tr>
<td>(b, d)</td>
<td>3 in start, 7 in loop</td>
</tr>
<tr>
<td>(b, i_0)</td>
<td>3 in start</td>
</tr>
<tr>
<td>(b, i_2)</td>
<td>3 in start</td>
</tr>
<tr>
<td>(c, d)</td>
<td>7 in loop, 7 before 6</td>
</tr>
<tr>
<td>(d, i_2)</td>
<td>7 in loop</td>
</tr>
</tbody>
</table>

7 in loop: overlap!
Compute code motions and overlaps

start:
0: i0 := 0
1: a := read()

loop:
2: i1 := φ(i0, i2)
3: b := a + 1
4: i2 := i1 + b
5: c := f(a)
6: compare i2 < c
8: blt loop

end:
7: d := i2 × 2
9: return d

Avoidable overlaps

<table>
<thead>
<tr>
<th>Pair</th>
<th>Overlapping placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, d</td>
<td>7 in loop</td>
</tr>
<tr>
<td>b, c</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, d</td>
<td>3 in start, 7 in loop</td>
</tr>
<tr>
<td>b, i0</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, i2</td>
<td>3 in start</td>
</tr>
<tr>
<td>c, d</td>
<td>7 in loop, 7 before 6</td>
</tr>
<tr>
<td>d, i2</td>
<td>7 in loop</td>
</tr>
</tbody>
</table>

7 not in loop: no overlap
Compute code motions and overlaps

```
start:
0:   i0 := 0
1:   a := read()
3:   b := a + 1
loop:
2:   i1 := \phi(i0, i2)
4:   i2 := i1 + b
5:   c := f(a)
6:   compare i2 < c
7:   d := i2 × 2
8:   blt loop
end:
9:   return d
```

Avoidable overlaps

<table>
<thead>
<tr>
<th>Pair</th>
<th>Overlapping placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, d</td>
<td>7 in loop</td>
</tr>
<tr>
<td>b, c</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, d</td>
<td>3 in start, 7 in loop</td>
</tr>
<tr>
<td>b, i0</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, i2</td>
<td>3 in start</td>
</tr>
<tr>
<td>c, d</td>
<td>7 in loop, 7 before 6</td>
</tr>
<tr>
<td>d, i2</td>
<td>7 in loop</td>
</tr>
</tbody>
</table>

3 in start, 7 in loop: overlap!
Compute code motions and overlaps

start:
0: i0 := 0
1: a := read()
loop:
2: i1 := \phi(i0, i2)
3: b := a + 1
4: i2 := i1 + b
5: c := f(a)
6: compare i2 < c
7: d := i2 \times 2
8: blt loop
end:
9: return d

Avoidable overlaps

<table>
<thead>
<tr>
<th>Pair</th>
<th>Overlapping placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, d</td>
<td>7 in loop</td>
</tr>
<tr>
<td>b, c</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, d</td>
<td>3 in start, 7 in loop</td>
</tr>
<tr>
<td>b, i0</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, i2</td>
<td>3 in start</td>
</tr>
<tr>
<td>c, d</td>
<td>7 in loop, 7 before 6</td>
</tr>
<tr>
<td>d, i2</td>
<td>7 in loop</td>
</tr>
</tbody>
</table>

3 not in start: no overlap
Compute code motions and overlaps

```
start:
0: i0 := 0
1: a := read()
loop:
2: i1 := φ(i0, i2)
3: b := a + 1
4: i2 := i1 + b
5: c := f(a)
6: compare i2 < c
7: d := i2 × 2
end:
8: blt loop
9: return d
```

### Avoidable overlaps

<table>
<thead>
<tr>
<th>Pair</th>
<th>Overlapping placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, d</td>
<td>7 in loop</td>
</tr>
<tr>
<td>b, c</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, d</td>
<td>3 in start, 7 in loop</td>
</tr>
<tr>
<td>b, i0</td>
<td>3 in start</td>
</tr>
<tr>
<td>b, i2</td>
<td>3 in start</td>
</tr>
<tr>
<td>c, d</td>
<td>7 in loop, 7 before 6</td>
</tr>
<tr>
<td>d, i2</td>
<td>7 in loop</td>
</tr>
</tbody>
</table>

7 not in loop: no overlap
Register allocation

Conflict graph with special edges for avoidable overlaps. Allocate to different registers if possible.

\[ \Rightarrow \]
Register allocation

Conflict graph with special edges for **avoidable** overlaps. Allocate to different registers **if possible**.

5 registers: easy allocation
Register allocation

Conflict graph with special edges for **avoidable** overlaps. Allocate to different registers **if possible**.

4 registers: place instruction 7 in block end to avoid overlaps
Register allocation

Conflict graph with special edges for avoidable overlaps. Allocate to different registers if possible.

3 registers: place 3 in loop and 7 in end
Integrate code motion and register allocation by letting the allocator choose necessary code motions.

Execution time improved by up to 4% 😊

...but no improvement on average 😞

**Conclusion:** Code motion is important, but simple heuristics suffice in practice.
Results

- Integrate code motion and register allocation by letting the allocator choose necessary code motions.
- Execution time improved by up to 4% 😊
- ...but no improvement on average 😞

Conclusion: Code motion is important, but simple heuristics suffice in practice.

Thank you!

This work was supported by the Austrian Science Fund (Fonds zur Förderung der wissenschaftlichen Forschung) under contract P21842, *Optimal Code Generation for Explicitly Parallel Processors.*