Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Talen en Compilers

2023 - 2024

David van Balen

Department of Information and Computing Sciences
Utrecht University

2024-01-23

14. Compiler optimizations

Announcements

- Grades soon
- Next lecture: summary
- Send topics, questions, example exercises to Lawrence

This lecture

Compiler optimizations

- Optimization passes
- Simple optimizations
- Loop optimizations
- Other optimizations

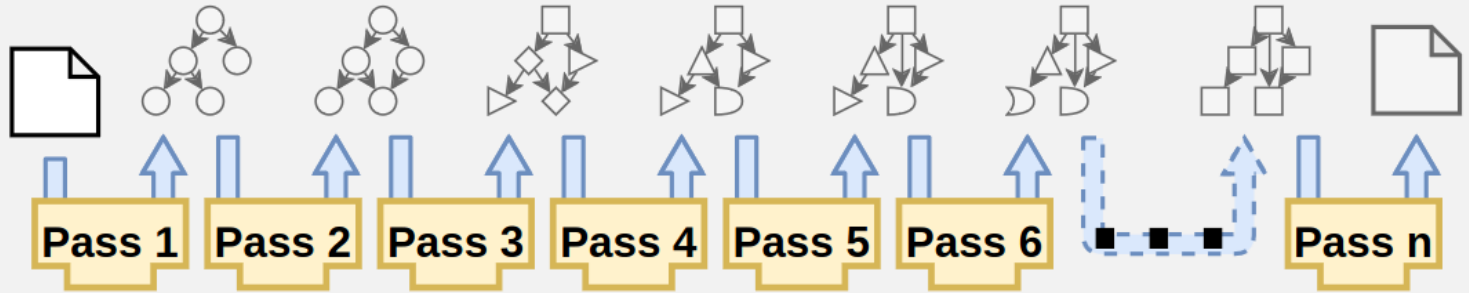
14.1 Optimization passes

What is a compiler optimization

- A terrible name
- Semantics-preserving code transformation
- Hopefully improving the code by some metric

Visualizing optimization passes

Typically the same input and output type, like pass 2.



14.2 Simple optimizations

Peephole optimizations

- Group of simple but effective optimizations
- Find and replace
- Usually on low-level instructions

Peephole optimizations

- Group of simple but effective optimizations
- Find and replace
- Usually on low-level instructions

- $x * 2 \Rightarrow x << 1$
- $x * 0 \Rightarrow 0$
- $x \leftarrow 3; x \leftarrow 4 \Rightarrow x \leftarrow 4$

Unreachable/dead code elimination

- Uncalled methods/functions
- Code after a return statement
- Patterns that cannot be matched
- ...

Tail call elimination

```
int add (int m, int n) {
    if (m = 0) then
        return n;
    else
        return add (m − 1, n + 1);
}
```

```
int add (int m, int n) {
    while (m != 0) {
        m = m − 1;
        n = n + 1;
    }
    return n;
}
```

14.3 Loop optimizations

Loop optimizations

- Loop unrolling
- Loop invariant code motion
- Loop fusion
- Loop fission

Loop unrolling

```
for (int i = 0; i < n; i++)
{doStuff (i);
}
```

```
for (int i = 0; i < n - 4; i += 4)
{doStuff (i);
  doStuff (i + 1);
  doStuff (i + 2);
  doStuff (i + 3);
}
```

Loop unrolling

```
for (int i = 0; i < n; i++)
{doStuff (i);
}
```

```
for (int i = 0; i < n - 4; i += 4)
{doStuff (i);
  doStuff (i + 1);
  doStuff (i + 2);
  doStuff (i + 3);
}
```

If n is not divisible by 4, you need to do extra iterations before or after the loop.

Loop invariant code motion

```
for (int i = 0; i < n; i++)
{x = 10 * y + cos (0.5);
  doStuff (i, x);
}
```

```
x = 10 * y + cos (0.5);
for (int i = 0; i < n; i++)
{doStuff (i, x);
}
```

Loop fusion

```
for (int i = 0; i < n; i++)
{doStuff1 (i);
}
for (int i = 0; i < n; i++)
{doStuff2 (i);
}
```

```
for (int i = 0; i < n; i++)
{doStuff1 (i);
  doStuff2 (i);
}
```

Loop fission

```
for (int i = 0; i < n; i++)
{doStuff1 (i);
  doStuff2 (i);
}
```

```
for (int i = 0; i < n; i++)
{doStuff1 (i);
}
for (int i = 0; i < n; i++)
{doStuff2 (i);
}
```

Loop fission

```
for (int i = 0; i < n; i++)
{doStuff1 (i);
  doStuff2 (i);
}
```

```
for (int i = 0; i < n; i++)
{doStuff1 (i);
}
for (int i = 0; i < n; i++)
{doStuff2 (i);
}
```

The opposite of fusion: Sometimes one is better, sometimes the other!

To choose, we might want to add analyses to our compiler.

14.4 Other optimizations

Inlining

$$\textbf{let } x = 5 \\ \textbf{in } x * y + x$$

$$5 * y + 5$$

Common Subexpression Elimination

$$\cos\,(5\;\mathrm{x})\;/\;(1+\cos\,(5\;\mathrm{x}))$$

$$\begin{array}{l}\textbf{let}\;\mathrm{y}=\cos\,(5\;\mathrm{x})\\ \textbf{in}\;\mathrm{y}\;/\;(1+\mathrm{y})\end{array}$$

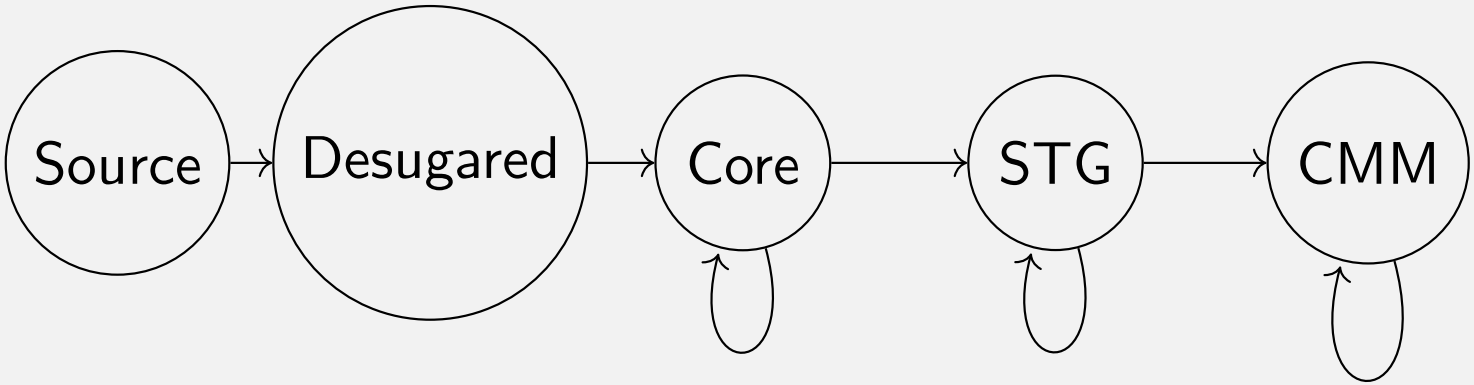
Common Subexpression Elimination

$$\cos\,(5\;\mathrm{x})\;/\;(1+\cos\,(5\;\mathrm{x}))$$

$$\textbf{let}\;\mathrm{y}=\cos\,(5\;\mathrm{x})\;\textbf{in}\;\mathrm{y}\;/\;(1+\mathrm{y})$$

Opposite of inlining: Tradeoff between computation and memory.

Compiler pipeline



My research

Analysis and code transformation

My research

Analysis and code transformation for optimal fusion

My research

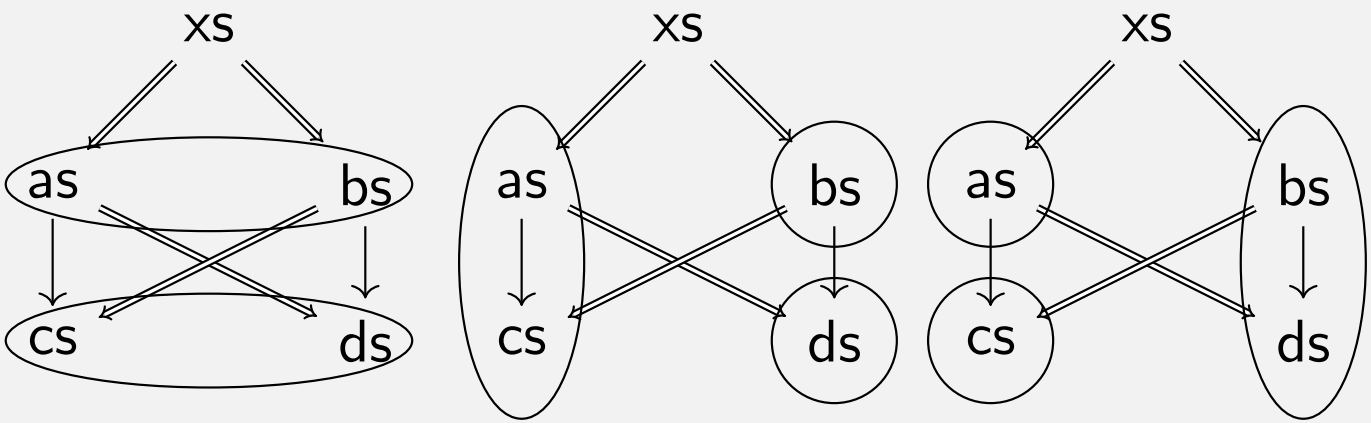
Analysis and code transformation for optimal fusion of array operations

My research

Analysis and code transformation for optimal fusion of array operations in a data parallel functional language

My research

Analysis and code transformation for optimal fusion of array operations in a data parallel functional language



Vertical fusion

```
for (int i = 0; i < n; i++) {
y [i] = 2 * x [i];
}
for (int i = 0; i < n; i++) {
z [i] = 4 + y [i];
}
return z;
```

```
for (int i = 0; i < n; i++) {
z [i] = 4 + 2 * x [i];
}
return z;
```

Replacing an array with a scalar

Eliminating n array reads and writes