Interference graph

Simone Campanoni
simonec@eecs.northwestern.edu
A graph-coloring register allocator structure

- Liveness analysis
- Interferences analysis
- Interference graph
- Code analysis
- Graph coloring
- Spill

\[ f \] with var spilled
\[ \text{spill}(f, \text{var}, \text{prefix}) \]

\[ f \] without variables and with registers

\[ f \]
Liveness analysis

**Goal:**
Identify the set of variables with values that will be used just before and just after a given instruction $i$, for every $i$ in a function $f$

```plaintext
(:myF 0 0
-
0  myVar1 <- 2
-
1  myVar2 <- 40
-
2  myVar3 <- myVar1
-
3  myVar3 += myVar2
-
4  rax <- myVar3
  return
)
```

IN (just before) and OUT (just after) sets

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>IN[0]</td>
<td>{}</td>
<td>myVar1</td>
<td>myVar3, myVar2</td>
<td>myVar3</td>
<td>myVar3</td>
</tr>
<tr>
<td>IN[1]</td>
<td>{}</td>
<td>myVar1</td>
<td>myVar3, myVar2</td>
<td>myVar3</td>
<td>myVar3</td>
</tr>
<tr>
<td>IN[2]</td>
<td>myVar1</td>
<td>myVar1, myVar2</td>
<td>myVar3, myVar2</td>
<td>myVar3</td>
<td>myVar3</td>
</tr>
<tr>
<td>IN[3]</td>
<td>myVar1</td>
<td>myVar1, myVar2</td>
<td>myVar3, myVar2</td>
<td>myVar3</td>
<td>myVar3</td>
</tr>
<tr>
<td>IN[4]</td>
<td>myVar1</td>
<td>myVar1, myVar2</td>
<td>myVar3, myVar2</td>
<td>myVar3</td>
<td>myVar3</td>
</tr>
</tbody>
</table>

Interference graph

- myVar1 → myVar2
- myVar3
The interference graph

- The Graph coloring algorithm assigns variables to registers
  
  myVar1 <- 5 → r10 <- 5

- This transformation must preserve:
  - The original code semantics
  - The constraints of the target architecture

- These constraints are encoded in the interference graph
- Nodes: variables
- Edges: interferences

- **Meaning of an edge:** 2 connected nodes must use different registers
Generating the interference graph

- 1 node per variable
- GP registers are considered variables
- Connect each pair of variables that belong to the same IN or OUT set
- Connect a GP register to all other registers (even those not used by \( f \))
- Connect variables in KILL[i] with those in OUT[i]
  - Necessary for dead code that defines a variable
- Handle constrained arithmetic via extra edges

\[
\begin{align*}
\emptyset & \quad \{ \text{myVar1} \} \\
\{ \text{myVar1} \} & \quad \{ \text{myVar1, myVar2} \} \\
\{ \text{myVar1, myVar2} \} & \quad \{ \text{myVar3, myVar2} \} \\
\{ \text{myVar3} \} & \quad \{ \text{myVar3} \}
\end{align*}
\]
Constrains in the target language L1

• The L1 instruction $x \text{ sop } sx$ is limited to only shifting by the value of $rcx$ (or by a constant)
• This must be encoded in the interference graph
• Add interference edges to disallow the illegal registers when building the interference graph
• For example, consider the following example:
  
a <<= b
we need to add edges between $b$ and every register except $rcx$
This ensures $b$ will end up in $rcx$ (or spilled)
A graph-coloring register allocator structure

- Liveness analysis
- Interferences analysis
- Interference graph

Register allocator
- Code analysis
- Graph coloring

Spill
- Spill(f, var, prefix)

f without variables and with registers

f with var spilled

IN, OUT