Interference graph

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

Liveness analysis

Interferences analysis

Interference graph

IN, OUT

Register allocator

Code analysis

Graph coloring

Spill

f with var spilled

spill(f, var, prefix)

f without variables and with registers
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
)
```

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

- $IN[0]$: {}
- $OUT[0]$: {}, $IN[1]$: {myVar1}

**Interference graph**

```
myVar1 ----> myVar2
  |        |
  v        v
  myVar3
```
The interference graph

• The Graph coloring algorithm assigns variables to registers
  myVar1 <- 5  \rightarrow  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
• Registers are considered variables
• Connect each pair of variables that belong to the same IN or OUT set
• Connect a register to all other registers (even those not used by \(f\))
• Connect variables in KILL\([i]\) with those in OUT\([i]\)
  • Unless it is \(x \leftarrow y\) where \(x\) and \(y\) are variables or registers
• Handle constrained arithmetic via extra edges

\[
\begin{align*}
&\{\} \\
&\{\text{myVar1}\} \\
&\{\text{myVar1}, \text{myVar2}\} \\
&\{\text{myVar3}, \text{myVar2}\} \\
&\{\text{myVar3}\} \\
&\text{r11} \quad \text{myVar1} \quad \text{myVar2} \\
&\text{r10} \quad \text{myVar3}
\end{align*}
\]
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:

  \[ a \ll b \]

  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

IN, OUT

Interference graph

- Code analysis
- Graph coloring

spill(f, var, prefix)

f with var spilled

Spill

f without variables and with registers