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 \( f \).

for every \( i \) in a function \( f \)

\[
(:\text{myF} 0 0
\]
0 \hspace{1cm} \text{myVar1} \leftarrow 2
1 \hspace{1cm} \text{myVar2} \leftarrow 40
2 \hspace{1cm} \text{myVar3} \leftarrow \text{myVar1}
3 \hspace{1cm} \text{myVar3} \leftarrow \text{myVar2}
4 \hspace{1cm} \text{rax} \leftarrow \text{myVar3}
return
\)

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

\[
\begin{align*}
\text{IN[0]} & : \{ \} \\
\text{OUT[0], IN[1]} & : \{ \text{myVar1} \} \\
\text{OUT[1], IN[2]} & : \{ \text{myVar1, myVar2} \} \\
\text{OUT[2], IN[3]} & : \{ \text{myVar3, myVar2} \} \\
\text{OUT[3], IN[4]} & : \{ \text{myVar3} \}
\end{align*}
\]

Interference graph

\[
\text{myVar1} \quad \text{myVar2} \quad \text{myVar3}
\]
The important of 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)
- And ...

Is this correct?

```plaintext
(:myF 0
 %MyVar1  <-  2
 %MyVar2  <-  40
 %MyVar3  <-  %MyVar1
 %MyVar4  <-  42
 %MyVar3  +=  %MyVar2
)

myVar1 —— myVar2 —— myVar3
   |       
   r10    r11

%r10  <-  2
%r11  <-  40
%r10  <-  %r10
%r11  <-  42
%r10  +=  %r11
print  %r10
```
Generating the interference graph (2)

- 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

\[
\begin{align*}
\text{myVar1} & \Rightarrow r10 \\
\text{myVar2} & \Rightarrow r11 \\
\text{myVar3} & \Rightarrow r10 \\
\text{myVar4} & \Rightarrow \text{spill}
\end{align*}
\]
Generating the interference graph (3)

• 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

```
(:myF 0
  %MyVar1 1
d  ;{myVar1}
  %MyVar2 2
d  ;{myVar1, myVar2}
  %MyVar2 0
d  ;{myVar1}
)
```

```
rcx = r11  myVar1
  \|  \\
  \|  \\
  \|  \\
  \|  \\
  \|  \\
  %r10 \\
  myVar2

MyVar1 => r10
MyVar2 => r11
```

Is this correct?
Constrains in the target language L1

• The L1 instruction `x 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)
Generating the interference graph (3)

• 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

```plaintext
(:myF 0
 %MyVar1 <- 1 ----------------- {myVar1}
 %MyVar2 <- 2 ----------------- {myVar1, myVar2}
 %MyVar2 <<= MyVar1 -------------- { }
)

rcx = r11  myVar1
  \  /  \\
   r10
  /  \\
myVar2

MyVar1 => rcx
MyVar2 => r11

(:myF 0
 %rcx <- 1
 %r11 <- 2
 %r11 <<= %rcx
)
```
A graph-coloring register allocator structure

- Liveness analysis
- Interferences analysis
  - IN, OUT
  - Interference graph

Register allocator:
- Code analysis
- Graph coloring
- Spill

- spill(f, var, prefix)
- f without variables and with registers
- f with var spilled

- f without variables and with registers
- f with variables and without registers
- function f
Homework #2

• Compute the interference graph of an L2 function given as input

```plaintext
(:myF
  0
  %myVar1 <- 5
  %myVar2 <- 0
  %myVar2 += %myVar1
  return
)
```

Your work needs to print to std::cout

```plaintext
test/interference/test1.L2f
```

• Implement the spiller (see Spilling.pdf)

```plaintext
myVar1 myVar2 r12 r13 r14 r15 rax rbp rbx
```

Nodes connected with the first one (the order between them doesn’t matter)

```plaintext
rsi r10 r11 r12 ... rbp rbx rcx rdi rdx
test/interference/test1.L2f.out
```

A node in the interference graph

The order between rows doesn’t matter
Testing the interference graph of your homework #2

• Under L2/tests/interference there are the tests you have to pass
• To test:
  • To check all tests: make test_interference
  • To check one test: ./interference test/interference/test1.L2f
• Check out each input/output for each test if you have doubts
  • test/interference/test1.L2f
  • test/interference/test1.L2f.out