Interference graph

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

Register allocator

Code analysis

Graph coloring

Spill

f without variables and with registers

IN, OUT

Liveness analysis

Interferences analysis

Interference graph

spill(f, var, prefix)
Outline

• What is the interference graph

• Algorithm to build the interference graph
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  
    (e.g., the second operand of the shift operation must be a constant or rcx)
  • These constraints are encoded in the interference graph

• Nodes: variables

• Edges: interferences

• **Meaning of an edge:** 2 connected nodes must use different registers
• Next we are going to learn the algorithm that automatically compute the interference graph

• The algorithm adds edges for different categories of constraints, one category at a time

• We will motivate each category of constraints by showing when the algorithm is incorrect if such category is not considered
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 ...

(`:`myF 0 0
 %myVar1 <- 2
 %myVar2 <- 40
 %myVar3 <- %myVar1
 %myVar4 <- 42
 %myVar3 += %myVar2
 print %myVar3
)

Graph coloring

(,:myF 0 0
 r10 <- 2
 r11 <- 40
 r10 <- r10
 r11 <- 42
 r10 += r11
 print r10
)

Is this correct?

Graph coloring
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

```plaintext
(:myF 0
  %myVar1 <- 2              •{ }
  %myVar2 <- 40              •{ %myVar1 }%myVar1
  %myVar3 <- %myVar1        •{ %myVar1, %myVar2 }%myVar1
  %myVar4 <- 42              •{ %myVar1, %myVar2 }%myVar2
  %myVar3 += %myVar2        •{ %myVar3, %myVar2 }%myVar3
  print %myVar3             •{ %myVar3 }
)

r10 | %myVar1

%myVar2

%myVar4 | %myVar3

r11

r10

spill

(:myF 0 1
  r10 <- 2
  r11 <- 40
  r10 <- r10
  mem rsp 0 <- 42
  r10 += r11
  print r10
)
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

\[
\begin{align*}
(\text{:myF 0}) \quad & \%myVar1 \leftarrow 1 \quad \{ \%myVar1 \} \\
& \%myVar2 \leftarrow 2 \quad \{ \%myVar1, \%myVar2 \} \\
& \%myVar2 \ll\ll \%myVar1 \quad \{ \} \\
(\text{:myF 0 0}) \quad & \begin{align*}
& \text{rcx} \rightarrow r11 \\
& r10 \leftarrow 1 \\
& r11 \leftarrow 2 \\
& r11 \ll\ll r10 \\
\end{align*} \\
\%
\end{align*}
\]

Is this correct?
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)
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

\[
\begin{align*}
\text{:myF 0} & \quad \%myVar1 \leftarrow 1 \quad \%myVar2 \leftarrow 2 \\
\text{myF 0} & \quad \%myVar2 \leftarrow 1 \\
\text{myF 0} & \quad \%myVar2 \leftarrow 1 \\
\end{align*}
\]
A graph-coloring register allocator structure

- Register allocator
  - Code analysis
  - Graph coloring
  - Spill

- Liveness analysis
- Interferences analysis

- IN, OUT

- Interference graph

- f without variables and with registers

- f with var spilled

- spill(f, var, prefix)
Homework #2

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

```cpp
:myF
0
%myVar1 <- 5
%myVar2 <- 0
%myVar2 += %myVar1
return
```

test/interference/test1.L2f

The order between rows doesn’t matter

%A node in the interference graph

%myVar1 %myVar2 r12 r13 r14 r15 rax rbp rbx

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

rsi r10 r11 r12 ... rbp rbx rcx rdi rdx

test/interference/test1.L2f.out

• Implement the spiller (see Spilling.pdf)

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