Graph coloring

Simone Campanoni
simonec@eecs.northwestern.edu
Outline

• Graph coloring

• Heuristics

• L2c
Graph coloring task

- Input: the interference graph
- Output: the interference graph where each node has a color
- Task: Color the nodes in the graph such that connected nodes have different colors
- Abstraction: colors are registers
- After performing the graph coloring task: Replace L2 variables with the registers specified by the colors
A graph-coloring register allocator structure

Assign colors

Interference graph, f

Code generation

Interference graph colored, f

Register allocator

Code analysis

Graph coloring

f with var spilled

spill(f, var, prefix)

f without variables and with registers

Spill
Colors

• At design time of the register allocator:
  Map general purpose (GP) registers to colors

• The L1 (15) GP registers:
  rdi, rsi, rdx, rcx, r8, r9, rax, r10, r11, r12, r13, r14, r15, rbp, rbx

• Each register has one node in the interference graph
  • Pre-colored nodes

• Before starting coloring the nodes related to variables:
  Color register nodes with their own colors
A coloring algorithm

Algorithm:
1. Repeatedly select a node and remove it from the graph, putting it on top of a stack
2. When the graph is empty, rebuild it
   • Select a color on each node as it comes back into the graph, making sure no adjacent nodes have the same color
   • If there are not enough colors, the algorithm fails
     • Spilling comes in here
     • Select the nodes you want to spill
Coalescing: the potential problem

(:myF 3 0
 %v0 <- rdi
 %v0 += rdi
 %v0 += rsi
 %v0 += r10
 %v1 <- %v0
 %v2 <- %v0
 rax <- %v0
 rax += %v1
 rax += %v2
 return
)

:myf(%p0, %p1, %p2){
   return (%p0 *2 + %p1 + %p2) * 3
}

We just need 1 register

No spilling necessary 😊

We need 3 registers 😞
Outline

• Graph coloring

• Heuristics

• L2c
Heuristics

• You need to decide the heuristics to use

• Next slides describe simple heuristics you can implement

• We will see more advanced heuristics later
  • You don’t have to implement them
  • But if you do:
    • your L2 compiler will generate more performant code
  • At the end of this class: all final compilers will compete
A coloring algorithm

Algorithm:
1. Repeatedly select a node and remove it from the graph, putting it on top of a stack
2. When the graph is empty, rebuild it
   • Select a color on each node as it comes back into the graph, making sure no adjacent nodes have the same color
   • If there are not enough colors, the algorithm fails
      • Spilling comes in here
      • Select the nodes you want to spill
Heuristic: select the nodes to remove

Observation:
• Suppose G contains a node m with $< K$ adjacent nodes
• Let G’ be the graph G without m
• If G’ can be colored with K colors, then so can G

Heuristic:
• Remove the node with the most edges that’s smaller than then number of colors (15 in L1)
• Remove the other nodes starting from the ones with more edges
A coloring algorithm

Algorithm:

1. Repeatedly select a node and remove it from the graph, putting it on top of a stack

2. When the graph is empty, rebuild it
   - Select a color on each node as it comes back into the graph, making sure no adjacent nodes have the same color
   - If there are not enough colors, the algorithm fails
     - Spilling comes in here
     - Select the nodes you want to spill
Heuristic: select the color to use

Heuristic:
• Sort the colors at design time starting from caller save registers
• Use the lowest free color
A coloring algorithm

Algorithm:

1. Repeatedly select a node and remove it from the graph, putting it on top of a stack

2. When the graph is empty, rebuild it
   - Select a color on each node as it comes back into the graph, making sure no adjacent nodes have the same color
   - If there are not enough colors, the algorithm fails
     - Spilling comes in here
     - Select the nodes you want to spill
Heuristic: select the variables to spill

Observation:
• Every time you spill:
  • Liveness analysis
  • Interference graph
  • Graph coloring

Heuristic:
• Add all nodes to the graph at step 2 of the algorithm
• Mark all nodes that represent variables that have no color
• Spill all variables represented by these marked nodes
Outline

• Graph coloring

• Heuristics

• L2c
Register allocator

Graph coloring

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

Code analysis

f without variables and with registers

f with var spilled

Spill

Your work

L1c

L1 program

L2 program

L2c

a.out
Homework #3: the L2 compiler

For every L2 function $f$,

1. Register allocator
2. L2 function $f$ with registers only
3. (stack-arg) translator
4. L2 function $f$ with registers only and without (stack-arg)
5. L1 function
Compiling and testing your L2 compiler

• Under L2/tests there are the L2 programs you’ll translate
• Build your L1 compiler:
  • Keep your L1 compiler sources in L1/src
  • Compile your L1 compiler: cd L1 ; make
• Build your L2 compiler:
  • Build your homework #2 under L2/src
  • Write new code to complete the translation from L2 to L1 in L2/src
  • Compile your L2 compiler: cd L2 ; make
• To test: cd L2 ; make test