Advanced graph coloring

Simone Campanoni
simone.campanoni@northwestern.edu
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 (variables) you want to spill
Outline

• Coalescing and freezing

• Advanced register order

• Advanced spilling
Limitation of our basic approach

(@myF 1
  %v0 <- rdi
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)

What is the best L1 code?

(@myF 1 0
  rax <- rdi
  rax += rdi
  rax += rdi
  return
)

SPILL!
Advanced heuristic: coalescing

(@myF 1
  %v0 <- rdi
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)

Are they useful?
(@myF 1 0
  rdi <- rdi
  rdi <- rdi
  r10 <- rdi
  rax <- rdi
  rax += rdi
  rax += r10
  return
)
Advanced heuristic: coalescing

(@myF 0
  %v0 <- rdi
  %v1 <- %v0
  %v2 <- %v0
  rax <- %v0
  rax += %v1
  rax += %v2
  return
)

(@myF 1 0
  r10 <- rdi
  rax <- rdi
  rax += rdi
  rax += r10
  return
)
Advanced heuristic: coalescing

(@myF 0
    %v0 <- rdi
    %v1 <- %v0
    %v2 <- %v0
    rax <- %v0
    rax += %v1
    rax += %v2
    return
)

(@myF 1 0
    rax <- rdi
    rax += rdi
    rax += rdi
    return
)
Coalescing problem

• Coalescing can significantly increase the quality of the code
• Merging N nodes increases the degree of the resulting node
• This might generate a graph that requires more colors
  • More spills!
Coalescing: the potential problem

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

• Graph coloring without coalescing succeeded!
• Let’s try to do coalescing before graph coloring
Coalescing: the potential problem

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

FAIL
Coalescing problem

• Coalescing can significantly increase the quality of the code
• Merging N nodes increases the degree of the resulting node
• This might generate a graph that requires more colors
  • More spills!
• So when should we apply it?
• Two common conservative strategies:
  1. Briggs
  2. George
Nodes $a$ and $b$ can be coalesced if the resulting node $ab$ will have fewer than $K$ neighbors of degree $\geq K$

- $K =$ Number of general purpose registers

- This coalescing is guaranteed not to turn a $K$-colorable graph into a non-$K$-colorable graph
Nodes $a$ and $b$ can be coalesced if for every adjacent node $t$ of $a$, either:

- $(t, b)$ already exists or
- $\text{Degree}(t) < K$
Graph coloring without coalescing

- Code analysis
  - Interference graph, f
- Simplify graph
- Select
- Spill
Graph coloring with coalescing

- Code analysis
  - Interference graph, \( f \)
  - Tag nodes to be move-related
  - Simplify graph only for not-move-related nodes with degree < GP registers
  - Coalesce with Briggs or George (Simplify not-move-related nodes)
  - Simplify all and Select
  - Spill
Advanced heuristic: freeze move nodes

1. Tag nodes to be move-related
2. Simplify graph only for not-move-related nodes with degree < GP registers
3. Coalesce with Briggs or George (Simplify not-move-related nodes)
4. Freeze (give up coalescing some nodes)
5. Simplify all and Select
Outline

• Coalescing and freezing

• Advanced register order

• Advanced spilling
Example

(@myF
  1
  %myV1 <- 1
  %myV2 <- 1
  %myV3 <- 1
  %myV4 <- 1
  %myV5 <- 1
  %myV6 <- 1
  %myV7 <- 1
  mem rdi 0 <- %myV1
  mem rdi 8 <- %myV2
  mem rdi 16 <- %myV3
  mem rdi 24 <- %myV4
  mem rdi 32 <- %myV5
  mem rdi 40 <- %myV6
  mem rdi 48 <- %myV7
  return
)

### Registers

<table>
<thead>
<tr>
<th>Arguments</th>
<th>Result</th>
<th>Caller save</th>
<th>Callee save</th>
</tr>
</thead>
<tbody>
<tr>
<td>rdi</td>
<td>rax</td>
<td>r10</td>
<td>r12</td>
</tr>
<tr>
<td>rsi</td>
<td></td>
<td>r11</td>
<td>r13</td>
</tr>
<tr>
<td>rdx</td>
<td></td>
<td>r8</td>
<td>r14</td>
</tr>
<tr>
<td>rcx</td>
<td></td>
<td>r9</td>
<td>r15</td>
</tr>
<tr>
<td>r8</td>
<td></td>
<td>rax</td>
<td>rbp</td>
</tr>
<tr>
<td>r9</td>
<td></td>
<td>rcx</td>
<td>rbx</td>
</tr>
</tbody>
</table>
Example

(@myF
  1
  %myV1 <- 1
  %myV2 <- 1
  %myV3 <- 1
  %myV4 <- 1
  %myV5 <- 1
  %myV6 <- 1
  %myV7 <- 1

Caller save
    r10
    r11
    r8
    r9
    rcx
    rdi
    rdx
    rsi
    rax

mem rdi 0 <- %myV1
mem rdi 8 <- %myV2
mem rdi 16 <- %myV3
mem rdi 24 <- %myV4
mem rdi 32 <- %myV5
mem rdi 40 <- %myV6
mem rdi 48 <- %myV7
return
)

Will we color this graph without spilling?
Yes
Example 2

(@myF
  1
  %myV1 <- 1
  %myV2 <- 1
  %myV3 <- 1
  %myV4 <- 1
  %myV5 <- 1
  %myV6 <- 1
  %myV7 <- 1

  mem rdi 0 <- %myV1
  mem rdi 8 <- %myV2
  mem rdi 16 <- %myV3
  mem rdi 24 <- %myV4
  mem rdi 32 <- %myV5
  mem rdi 40 <- %myV6
  mem rdi 48 <- %myV7

  return
)

• Will we color this graph without spilling?
• Can we do better?
• What about using callee save registers?
  • Yes, but we need to save them at the beginning of the function and restore them before every return

Call @myF2 0
:ret
mem rsp -8 <- :ret

Example: assuming 2 caller save registers

Approach: advanced graph coloring

```plaintext
(@myF   %myV1 -> rsi
  1       %myV2 -> r12

rsi   %myV1 <- 1
r12   %myV2 <- 3

mem rdi 0  <- %myV2
mem rdi 8  <- %myV1

return)

... // computation that uses myV* variables
```
Example: assuming 2 caller save registers

Approach: advanced graph coloring

```plaintext
(@myF               %myV1 -> rsi
  1 1
mem rsi 0 <- rsi
%myV1 <- 1
rsi %myV1 -> rsi
r12               %myV2 -> r12
%myV2 <- 3
mem rdi 0 <- %myV2
mem rdi 8 <- %myV1
r12 <- mem rsp 0
return)

... // computation that uses myV* variables
```
Example: assuming 2 caller save registers

<table>
<thead>
<tr>
<th>Basic solution</th>
<th>L2</th>
<th>Advanced solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>(@myF %myV1 -&gt; Spill %myV2 -&gt; rsi)</td>
<td>(@myF 1 1)</td>
<td>(@myF %myV1 -&gt; rsi %myV2 -&gt; r12)</td>
</tr>
<tr>
<td>1 1</td>
<td></td>
<td>1 1</td>
</tr>
<tr>
<td>rsi &lt;- 1</td>
<td></td>
<td>mem rsp 0 &lt;- r12</td>
</tr>
<tr>
<td>mem rsp 0 &lt;- rsi</td>
<td></td>
<td>rsi &lt;- 1</td>
</tr>
<tr>
<td>rsi &lt;- 3</td>
<td></td>
<td>r12 &lt;- 3</td>
</tr>
<tr>
<td>... // computation</td>
<td></td>
<td>... // computation</td>
</tr>
<tr>
<td>mem rdi 0 &lt;- rsi</td>
<td></td>
<td>mem rdi 0 &lt;- r12</td>
</tr>
<tr>
<td>rsi &lt;- mem rsp 0</td>
<td></td>
<td>mem rdi 8 &lt;- rsi</td>
</tr>
<tr>
<td>mem rdi 8 &lt;- rsi</td>
<td></td>
<td>r12 &lt;- mem rsp 0</td>
</tr>
<tr>
<td>return</td>
<td></td>
<td>return</td>
</tr>
<tr>
<td>)</td>
<td>)</td>
<td>)</td>
</tr>
</tbody>
</table>

Why is it worth it?
Select

Basic select (Graph_coloring.pdf slides)

Fail: You can only select a callee-save register if it has not already been used in the function.

Spill or save a callee save register?

Success

Spill

Modify f to save/restore a callee save register

Restart w/o spill

You can only select a callee-save register if it has not already been used in the function.

You can only select a callee-save register if it has not already been used in the function.
Advanced heuristics: register order

• Until now:
  • Caller-save registers are used first
  • Callee-save registers are used only at the end

• Change the order of registers depending on the code in $f$
  • E.g., a lot of calls => prefer callee save registers
  • E.g., a few calls => prefer caller save registers

• This heuristic requires extra code analysis to count #calls
Advanced heuristic: node selection

• Idea: variables used the most at run-time should be in registers

• Approach: give priority to nodes (variables) used in loops

• This heuristic requires a code analysis usually found in middle-ends: loop identification
Outline

• Coalescing and freezing

• Advanced register order

• Advanced spilling
Advanced heuristic: spilling

• Spill a subset of variables at every iteration
  • E.g., 1 at a time

• After having spilled variables
  • Run the register allocation algorithm for spilled variables
  • This will save space in the stack (lower memory pressure)
  • 1 color = 1 stack location
Always have faith in your ability

Success will come your way eventually

Best of luck!