Iterated Register Coalescing

LAL GEORGE
ANDREW W. APPEL
Background

• Register allocation
  • Register: store variables temporarily to speed up program
  • Limitation: too expensive and limited number

• What we have learned in last semester?
  • CS314
  • $A = b + c \quad r_1 \leq r_2 + r_3$
  • Each copy instruction means an allocation for registers
  • Too aggressive

• Question:
  • How to allocate registers more efficiently?
How to allocate registers more efficiently?

• Easy: Sometimes the copy instruction is not necessary
  • A = 0 * c; a = 0 + b;

• Advanced: Sometimes we can eliminate copy instructions by put them together ---- coalesce registers
  • a = b

• Then how to coalesce registers to eliminate copy instructions?
  • *Iterated Register Coalescing*
Abstract

• How to eliminate copy instructions by target registers in register allocator?
  • Graph-coloring register allocation: *simplification*
  • If source and destination don’t interfere, then *coalesce*
    • Chaitin: too aggressive, make a graph uncolorable
    • Briggs: too conservative, leaving too many move instructions

• Contribution:
  • A interleave coloring reduction algorithm using *simplification and coalescing*
    based on Briggs algorithm but more aggressive
What register allocation is important?

• Registers are expensive

• Compilers generate large number of move instructions
  • Static Single-Assigment(SSA) compilers: phi function
  • Non SSA based compilers: formal parameters to fresh temporaries

• These move instructions lead to copy propagations and produce a graph with temporaries

• Interleaving Chaitin-style with Beiggs-style together and get better performance
What is graph coloring?

• Color the vertices by different colors and the connected vertices do not have same color.
  • Application: map coloring

• Graph coloring & Register allocation
  • Each node represent a datum/live range/temporary
  • Each edge represent if two data are live at the same time (*interfere*)
  • After coloring, each color represents an allocation of a register
  • The number of colors available is equal to the number of registers.
How to allocate registers?

• K-coloring is NP-complete, then we use a approximation algorithm
• 1) build interference graph
• 2) coalesce: remove unnecessary move instructions (i.e. source and destination do not have an edge). Keep doing it.
• 3) Simplify: Color graph by K colors.
  • Remove node m with fewer than K neighbors
  • Keep doing it
• 4) Spill: If all nodes with >= K neighbors (significant degree). Choose a node, remove it from graph and put it into memory
• 5) Select: assign colors to nodes in the graph. Keep adding nodes and coloring nodes to an empty graph.
Example

Fig. 2. Interference graph. Dotted lines are not interference edges but indicate move instructions.
Example

• Graph from RTOS Lab
Coalescing – previous work

- Eliminate redundant move instructions with an interference graph
- Chaitin coalesced any pair of nodes not connected by an interference edge
  - Too aggressive, unite too many edges to a node, may not be colorable after this reckless coalescing
  - It is worse for precolored register allocation, as limited colors
- Briggs proposed a conservative coalescing strategy: coalescing node should have less than $K$ neighbors of significant degree
  - Conservative, as coalescing node has more than $K$ neighbors of significant degree but two of these neighbors might get same color, which means the graph is still colorable
Pros and Cons of Briggs’ algorithm

• Pros: Conservative coalescing is successful at removing move instructions without spills
• Cons: some move instructions remain

• Solution:
  • Briggs proposed biased coloring heuristic during select phase for nodes with more than K neighbors of significant degree
  • X <= Y : give Y same color as X; or give X same color as Y
Example of biased coloring

- 4 registers in total
- Coalescing b and j
- $E d k m$ are significant-degree neighbors
- 4 neighbors $\geq$ 4 registers

- Make j and b having same color
But...biased selection success by chance

• Both f and b are r2
• Then j can’t be r2
• Then the movement between j and b is essential

• We need to look ahead when assigning for f
• But looking ahead is expensive
Iterated register coalescing

• Chaitin’s simplification + Briggs’ conservative coalescing

• Loop of simplify and coalesce
example

• B c d j are move related nodes
• 1) initial worklist to simplify: non-move related nodes. i.e. g h f
• 2) do simplify on g h f
• 3) coalescing: c d, as they only have two neighbors of significant degree, m and b
• 4) coalescing: b j, as they only have two neighbors of significant degree, m and e
• 5) simplify again....
Fig. 6. Interference graph after coalescing d and c.

Fig. 7. Interference graph after coalescing b and j.
Graph coloring implementation

• Data structure: adjacency list
  • Derive a list of neighbors that have been colored
  • Two adjacency lists are unioned during coalescing
  • Much smaller than bit matrix

• Four list
  • Low-degree non-move-related nodes (*simplifyWorklist*)
  • Coalesce candidates: move-related nodes that have not been proved uncoalesceable (*worklistMoves*)
  • Low-degree move-related nodes (*freezeWorklist*)
  • High-degree nodes (*spillWorklist*)

• Precolored machine registers that do not interference, paint them same color
Benchmarks

• Single module
• Multiple modules
  • Function call modules
  • Large number of live ranges

• Live range shows the number of nodes in the interference graph
• Average degree indicate the average length of adjacency lists
Result

- Spill statics
  - The number of store and fetch instructions is almost equal to nodes spilled
  - Spilled nodes may only have one definition and use

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Nodes spilled</th>
<th>Instructions</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>store</td>
</tr>
<tr>
<td>knuth-bendix</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>vboyer</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>yacc.sml</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>utils.sml</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>yacc.grm.sml</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>nucleic</td>
<td>701</td>
<td>701</td>
</tr>
<tr>
<td>simple</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>format</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>ray</td>
<td>6</td>
<td>6</td>
</tr>
</tbody>
</table>

Fig. 10. Spill statistics.
Result

• One round and Iterated algorithms
  • Coalesced are removed moves using conservative coalescing strategy
  • Constrained are moves become constrained after adding interference edge during coalescing
  • Biased are moves coalesced using biased selection
  • Frozen are moves can not be coalesced using biased selection
Result

• One round
  • 62% moves are removed

• Iterated scheme
  • Biased does not apply
  • 84% moves are removed
Results

• Code size
  • 5% smaller size
  • while 4.4% faster because of better I-cache performance although moves are fast kind of instructions
Results

- Six callee-save registers vs. three callee-save registers
- More faster

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>3 callee-save</th>
<th>6 callee-save</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>knuth-bendix</td>
<td>7.06 sec</td>
<td>6.99</td>
<td>1 %</td>
</tr>
<tr>
<td>vboyer</td>
<td>2.40</td>
<td>2.30</td>
<td>4</td>
</tr>
<tr>
<td>mlyacc</td>
<td>3.50</td>
<td>3.18</td>
<td>9</td>
</tr>
<tr>
<td>simple</td>
<td>28.21</td>
<td>27.51</td>
<td>2</td>
</tr>
<tr>
<td>format</td>
<td>8.76</td>
<td>8.73</td>
<td>0</td>
</tr>
<tr>
<td>ray</td>
<td>47.20</td>
<td>44.34</td>
<td>6</td>
</tr>
</tbody>
</table>

Fig. 16. Execution time, varying the number of callee-save registers.
Conclusion

• A good register allocator with iterated simplify and coalesce phases
• It is very successful to move copy instructions and allocate limited registers

• Q&A
Rematerialization

• Constant values can be spilled very cheaply
  • No movement; just reload

• Constant propagation

```
int x = 14;
ext y = 7 - x / 2;
return y * (28 / x + 2);
```

Propagating x yields:

```
int x = 14;
ext y = 7 - 14 / 2;
return y * (28 / 14 + 2);
```

• If a and b are constant, coalesce
• If neither a nor b are constant, coalesce
• No reckless coalescing
  • Can not afford so many precolored nodes
• No constant-propagation

• Optimization are guided by the number of registers available
• Chaitin’s reckless coalescing produced too many spills
• Briggs’s conservative coalescing left too many move instructions
Pros

• Integrate copy propagation with register allocation to use registers more effectively without unnecessary moves or spills.
Two strategy in coloring

• Optimistic Coloring
  • If there are no low-degree nodes, put it on the stack instead of spill it.
  • Because Some neighbors may have same colors
• Only guarantee not to increase the number of potential spills
• Pessimistic Coloring

• Coalescing significantly reduces the number of temporaries and instructions in the graph which speed up the subsequent rounds of build and simplify.
• Chaitin’s pessimistic coloring: guarantee not to introduce new spills