Review: Chaitin-Briggs Style Allocator

1. Get live range & rename
2. Build interference graph
3. Copy coalescing: two variables associated by one MOV instruction can be coalesced
4. Calculate spilling cost for every live range
5. Choose a live range to spill if necessary
6. If possible: remove a node from the graph - if the number of neighbors < k
7. Select the node as a colorable node and push it to stack
8. Otherwise: choose a node to spill, remove the node and its incident edges. But still push the node into stack.
9. Insert spills and rebuild interference graph
10. Where to insert spills? Live range splitting can be seamlessly integrated here.
Linear Scan Allocator

- Coloring based allocator too expensive for Just-In-Time (JIT)
  * Compilation happens at runtime for JIT environment

- Linear-scan style allocator based on approximated interference graph
  * Similar to the bottom-up local allocation algorithm
  * Interference graph is an interval graph
  * Coloring an interval graph is polynomial-time problem
  * Algorithm does a “linear” scan of the graph
  * Linear scan produces faster, albeit less precise, allocators
Linear Scan Allocator

- Building the interference graph
  * Consider the procedure as a linear list of operations
  * A live range for some name is an interval (x,y)
  * x and y are the indices of two operations in the list, with x < y
  * Every operation where name is live falls between x & y, inclusive
    * Interval graph might overestimates interference

- The Algorithm
  * Use bottom-up local
  * Distance to next use is well defined
  * Algorithm is fast & produces reasonable allocations
  * Algorithm does a “linear” scan of the graph

A live range is a contiguous interval
Review: Bottom-up Local Allocation

- Bottom up (3 physical registers: ra, rb, rc)

<table>
<thead>
<tr>
<th>source code</th>
<th>life ranges</th>
<th>ra</th>
<th>rb</th>
<th>rc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 loadI 1028 ⇒ r1 // r1</td>
<td>r1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 load r1 ⇒ r2 // r1 r2</td>
<td>r1</td>
<td>r2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3 mult r1, r2 ⇒ r3 // r1 r2 r3</td>
<td>r1</td>
<td>r2</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>4 loadI 5 ⇒ r4 // r1 r2 r3 r4</td>
<td>r4</td>
<td>r2</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>5 sub r4, r2 ⇒ r5 // r1 r5 r3</td>
<td>r4</td>
<td>r5</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>6 loadI 8 ⇒ r6 // r1 r5 r3 r6</td>
<td>r6</td>
<td>r5</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>7 mult r5, r6 ⇒ r7 // r1 r7 r3</td>
<td>r6</td>
<td>r7</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>8 sub r7, r3 ⇒ r8 // r1 r8</td>
<td>r6</td>
<td>r8</td>
<td>r3</td>
<td></td>
</tr>
<tr>
<td>9 store r8 ⇒ r1 //</td>
<td>r1</td>
<td>r8</td>
<td>r3</td>
<td></td>
</tr>
</tbody>
</table>
The interference graph of a program in **pure SSA** form is a chordal graph \([1,2,3]\) 
* Recall that Chaitin-Briggs works from live ranges that are sets of SSA names
* Pure SSA based allocator, every raw name is a live range
* Allocate each live range, then insert copies for \(\Phi\)-function

A **chordal graph** is in which all cycles of four or more vertices have a chord

Chordal graphs can be **optimally colored** in \(O(N)\) time.
* Pure SSA allocator can allocate optimal number of registers in \(O(N)\) time
* Perfect elimination order
* Space efficient, but not necessarily computation efficient.
  — Non-trivial amount of copies if not colored appropriately.
SSA Allocator

- Register number is not a problem — always optimal
- Copy coalescing (note this concept appeared before)
SSA Allocator

- Register number is not a problem — always optimal
- Copy coalescing (note this concept appeared before)
SSA Allocator

- Register number is not a problem — always optimal
- Copy coalescing (note this concept appeared before)

So \( Rc, Ra, Rb \) can potentially be coalesced!

*Copy coalescing reduces the number of MOV instructions!* [4]
References

• [1] Pereira and Palsberg. “Register allocation via coloring of chordal graphs”. APLOS'05


• [3] [Brisk 2005] Philip Brisk and Foad Dabiri and Jamie Macbeth and Majid Sarrafzadeh, *Polynomial-Time Graph Coloring Register Allocation*, 14th International Workshop on Logic and Synthesis