15-745

Graph Coloring
Register Allocation

Intro to Global Register Allocation

Problem:
• Allocation of variables (pseudo-registers) to hardware registers in a procedure

One of the most important optimizations
• Memory accesses are more costly than register accesses
  – True even with caches
  – True even with CISC architectures
• Important for other optimizations
  – E.g., redundancy elimination assumes old values are kept in registers
• When it does not work well, the performance impact is noticeable.

Terminology

Allocation
• decision to keep a pseudo-register in a hardware register
• prior to register allocation, we assume an infinite set of registers
  – (aka “temps” or “pseudo-registers”).

Spilling
• when allocation fails...
  • a pseudo-register is spilled to memory, if not kept in a hardware register

Assignment
• decision to keep a pseudo-register in a specific hardware register

What are the Problems?

• For this example:
  • What is the minimum number of registers needed to avoid spilling?
  • Given \( n \) registers in a machine, is spilling necessary?
  • Find an assignment for all pseudo-registers, if possible.
  • If there are not enough registers in the machine, how do we spill to memory?
Abstraction for Reg Alloc & Assignment

Intuitively:

- Two pseudo-registers interfere if at some point in the program they cannot both occupy the same register.

Interference graph: an undirected graph, where

- nodes = pseudo-registers
- there is an edge between two nodes if their corresponding pseudo-registers interfere

Register Allocation and Coloring

- A graph is n-colorable if every node in the graph can be colored with one of n colors such that two adjacent nodes do not have the same color.

- Assigning n registers (without spilling) = Coloring with n colors
  - assign a node to a register (color) such that no two adjacent nodes are assigned the same register colors

- Is spilling necessary? = Is the graph n-colorable?

- To determine if a graph is n-colorable is NP-complete, for n>2
  - Too expensive
  - Heuristics

Simple Algorithm

Build an interference graph

- refining notion of a node
- finding the edges

Coloring

- use heuristics to try to find an n-coloring
  - Success ⇔ colorable and we have an assignment
  - Failure ⇔ graph not colorable, or graph is colorable, but we couldn’t find a coloring

Nodes in an Interference Graph

- A = ...
  - IF A goto L1
- B = ...
  - = A
- D = B
- L1: C = ..
  - = A
  - D = C
- A = 2
- = A
Live Ranges & Merged Live Ranges

- **Motivation:** to create an interference graph that is easier to color
  - Eliminate interference in a variable’s “dead” zones.
  - Increase flexibility in allocation:
    - can allocate same variable to different registers

- A **live range** consists of a definition and all the points in a program (e.g. end of an instruction) in which that definition is live.
  - How to compute a live range?

- Two overlapping live ranges for the same variable must be merged

Merging Live Ranges

Merging definitions into equivalence classes:
- Start by putting each definition in a different equivalence class
- For each point in a program
  - if variable is live, and there are multiple reaching definitions for the variable
  - merge the equivalence classes of all such definitions into one equivalence class

From now on, refer to merged live ranges simply as live range
- Merged live ranges are also known as “webs”
**Edges of Interference Graph**

Intuitively:
- Two live ranges (necessarily of different variables) may interfere if they overlap at some point in the program.
- Algorithm:
  - At each point in program, enter an edge for every pair of live ranges at that point

An optimized definition & algorithm for edges:
- For each inst $i$
  - Let $x$ be live range of definition at inst $i$
  - For each live range $y$ present at end of inst $i$
  - Insert an edge between $x$ and $y$

- Faster
- Better quality?

\[
A = 2 \quad (A_2, D_{1,2})
\]

Edge between $A_2$ and $D_{1,2}$

**Example: Interference Graph**

So was it worth it to split the live ranges?

**Example 2**

**Coloring**

- Reminder: coloring for $n > 2$ is NP-complete
- Observations
  - A node with degree $< n$ implies it can always be colored successfully, given its neighbors' colors
  - A node with degree $= n$ implies
  - A node with degree $> n$ implies
### Coloring Algorithm

**Algorithm**
- Iterate until stuck or done
  - Pick any node with degree < n
  - Remove the node and its edges from the graph
- If done (no nodes left)
  - reverse process and add colors

**Example (n = 3)**

- Note: degree of a node may drop in iteration
- Avoids making arbitrary decisions that make coloring fail

### What Does Coloring Accomplish?

**Done:**
- colorable
- also obtained an assignment (colors correspond to registers)

**Stuck (n = 2):**
- colorable or not?

- One solution: optimistically remove nodes and hope we get lucky...

### Checkpoint

**Problems:**
- Given n registers in a machine, is spilling avoided?
- Find an assignment for all pseudo-registers, whenever possible.

**Solution:**
- Abstraction: an interference graph
  - nodes: (merged) live ranges
  - edges: presence of live range at time of definition
- Register Allocation and Assignment problems
  = n-colorability of interference graph
  ⇒ NP-complete
- Heuristics to find an assignment for n colors
  - **successful:** colorable, and finds assignment
  - **unsuccessful:** colorability unknown & no assignment

### Discussion

**What about when we can’t k-color?**
- spill to memory: next time

**Is the minimum coloring always what we want?**
- Hint: no

**What about architecture strangeness?**
- subword registers (x86, 68k, ColdFire...)
- register pairing (HP PA-RISC, SPARC, x86)
- register classes (x86, 68k, ColdFire...)

---

*Register Allocation © Seth Copen Goldstein & Todd C. Mowry & David Ryan Koes 2002-3*
An Improvement: Move Coalescing

Basic idea:
- eliminate moves by assigning the src and dest to the same register
- copy propagation and dead code elimination can’t eliminate all unnecessary moves

How can we modify our interference graph to do this?

An Exciting New Example

First compute live ranges...
...then construct interference graph

An Exciting New Example cont.

Want u and v to be assigned same color...
...merge u and v to form a single node

u and v are special:
A move whose source is not live-out of the move is a candidate for coalescing

That is, if the src and dest don’t interfere

Is Coalescing Always Good?

move edge

And the winner is?
2 colorable
3 colorable
When should we coalesce?

**Always**
- If we run into trouble start un-coalescing
  - no nodes with degree < k, see if breaking up coalesced nodes fixes
- yuck

**Only if we can prove it won’t cause problems**
- Briggs: Conservative Coalescing
- George: Iterated Coalescing

![Graph example](image)

When we simplify the graph, we remove nodes of degree < k...
want to make sure we will still be able to simplify coalesced node, uv

Briggs: Conservative Coalescing

- Can coalesce u and v if:
  - (# of neighbors of uv with degree ≥ k) < k
- Why?
  - Simplify pass removes all nodes with degree < k
  - # of remaining nodes < k
  - Thus, uv can be simplified

![Graph example](image)

What does Briggs say about k = 3? k = 2?

George: Iterated Coalescing

- Can coalesce u and v if
  - foreach neighbor t of u
    - t interferes with v, or, doesn’t change degree of t < k
    - removed by simplification

**Resulting node uv will**
- have degree equal to degree of v

**Why?**
- Let S be set of neighbors of u with degree < k
- If no coalescing, simplify removes all nodes in S, call that graph G¹
- If we coalesce u we can still remove all nodes in S, call that graph G²
- G² is a subgraph of G¹

![Graph example](image)

No coalescing, after simplification

After coalescing and simplification
### Why Two Methods?

- Why not?
- With *Briggs*, one needs to look at all neighbors of \(a\) & \(b\)
- With *George*, only need to look at neighbors of \(a\).

So:

- Use *George* if one of \(a\) & \(b\) has very large degree
- Use *Briggs* otherwise

### Where We Are

1. **Build**
2. **Simplify**
3. **Coalesce**

### Where We’re Going

1. **Build**
2. **Simplify**
3. **Coalesce**
4. **Potential Spill**
5. **Select**
6. **Actual Spill**

plus a bunch of important details...

---

**Proto-Professor Algarth Zag, pioneer in fire research**

Hmm...Interesting...Now, Thog, next YOU go in there and see where all my other research students have gotten to......