Register Allocation
Register Allocation

- Introduction
- Problem Formulation
- Algorithm
Register Allocation Goal

- Allocation of variables (pseudo-registers) in a procedure to hardware registers
- Directly reduces running time by converting memory access to register access
  - What’s memory latency in CPU cycles?
Example

\[ a = b + c \]

How long will the loop take, if \( a, b, \) and \( c \) are all in memory?
Example

\[ a = b + c \]

\[ \text{Ld } r1, b \]
\[ \text{Ld } r2, c \]
\[ \text{Add } r0, r1, r2 \]
\[ \text{St } a, r0 \]
Example

Ld r1, b
Ld r2, c
Add r0, r1, r2
St a, r0

How long will the loop take?
Revisit SSA Example

- Mapping $a_i$ to $a$ is like register allocation

\[
\begin{align*}
& a_1 = b_1 \\
& a_1 + 5 \\
& b_2 = \\
& a_1 + 5 \\
& a_1 = b_1 \\
& b_1 + 5 \\
& b_2 = \\
& a_1 + 5 \\
& a_1 = b_1 \\
& b_1 + 5 \\
& b_2 = \\
& b_1 + 5
\end{align*}
\]
High-level Steps

- Find an assignment for all pseudo-registers or alias-free variables.
- Assign a hardware register for each variable.
- If there are not enough registers in the machine, choose registers to spill to memory.
Register Allocation

- Introduction
- Problem Formulation
- Algorithm
Problem Formulation

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

- Interfere Graph:
  - nodes = pseudo-registers
  - There is an edge between two nodes if their corresponding pseudo-registers interfere
Pseudo-registers

\begin{align*}
a &= 1 \\
b &= 2 \\
c &= a + b \\
d &= a + 3 \\
e &= a + b
\end{align*}

\begin{align*}
t1 &= 1 \\
t2 &= 2 \\
t3 &= t1 + t2 \\
t4 &= t1 + 3 \\
t5 &= t1 + t2
\end{align*}
Interference Graph

- $t_1$
- $t_2$
- $t_3$
- $t_4$
- $t_5$
Graph Coloring

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

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

- How many colors are needed?
Graph Coloring and Register Allocation

- 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 registers (colors)
- Is spilling necessary? = Is the graph $n$-colorable?
Register Allocation

- Introduction
- Problem Formulation
- *Algorithm*
Algorithm

- Step 1: Build an interference graph
  - Refining the notion of a node
  - Finding the edges

- Step 2: Coloring
  - Use heuristics to find an n-coloring
    - Successful → colorable and we have an assignment
    - Failure → graph not colorable, or graph is colorable, but it is too expensive to color
Step 1a: Nodes in Interference Graph

Every pseudo-register is a node
Step 1b: Edges of Interference Graph

- Intuition
  - Two live ranges (necessarily different variables) may interfere if they overlap at some point in the program
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 the 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.
Two overlapping live ranges for same variable must be merged

\[ a = b \]
\[ a = c \]
\[ a = a + d \]
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 ranges
Algorithm for Edges

- **Algorithm**
  - For each instruction $i$
    - Let $x$ be live range of definition at instruction $i$
    - For each live range $y$ present at end of instruction $i$
      - Insert an edge between $x$ and $y$
Example

\[ b = \]
\[ d = a \text{ (d2)} \]
\[ = b + d \]

\[ c = \]
\[ d = a \text{ (d1)} \]
\[ = d + c \]

\[ a = d \text{ (a2)} \]
\[ = a \]

doesn’t use a, b, c or d

reachings def
\{a1\}
\{a1,c\}
\{a1,c,d1\}
\{a1,c,d1\}

liveness
\{a\}
\{a,c\}
\{c,d\}
\{d\}
Interference Graph

\[ t_1 = 1 \]
\[ t_2 = 2 \]
\[ t_3 = t_1 + t_2 \]
\[ t_4 = t_1 + 3 \]
\[ t_5 = t_1 + t_2 \]

\[ t_1 \rightarrow t_2 \rightarrow t_3 \rightarrow t_4 \rightarrow t_5 \]

\[ t_1 - t_5 \text{ are not live} \]
Interference Graph

t1 = 1

t2 = 2

t3 = t1 + t2

t4 = t1 + 3

t5 = t1 + t2

t1-t5 are not live


Interference Graph

t1 = 1

t2 = 2

t3 = t1 + t2

t4 = t1 + 3

t5 = t1 + t2

t1-t5 are not live
Step 2: Coloring

- Reminder: coloring for n>2 is NP-complete
- Observations
  - A node with degree < n?
  - A node with degree = n?
  - A node with degree > n?
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
- **Note**: degree of a node may drop in iteration
Colorable by 3 Colors?

\[ t_1 = 1 \]
\[ t_2 = 2 \]
\[ t_3 = t_1 + t_2 \]
\[ t_4 = t_1 + 3 \]
\[ t_5 = t_1 + t_2 \]

\( t_1 \)-\( t_5 \) are not live
Colorable by 3 Colors?

Pick t5 and remove its edges
Colorable by 3 Colors?

Pick $t_4$ and remove its edges
Colorable by 3 Colors?

Pick t3 and remove its edges
Colorable by 3 Colors?

Pick t2 and remove its edges
Register Assignment

- Reverse process and add color different from all its neighbors
Register Assignment

- Color t2
Register Assignment

- Color t3
Register Assignment

- Color t4

```
t1/r1  t2/r2  t3/r3
   ↓    ↓    ↓
t4/r3  t2/r2  t3/r3
```

Stanford University

CS243 Winter 2006
Register Assignment

- Color t₅
After Register Allocation

\[
\begin{align*}
  t1 &= 1 \\
  t2 &= 2 \\
  t3 &= t1 + t2 \\
  t4 &= t1 + 3 \\
  t5 &= t1 + t2
\end{align*}
\]

\[
\begin{align*}
  r1 &= 1 \\
  r2 &= 2 \\
  r3 &= r1 + r2 \\
  r3 &= r1 + 3 \\
  r1 &= r1 + r2
\end{align*}
\]
When Coloring Fails

- Use heuristics to improve its chance of success and to spill code

- Algorithm
  - Iterate until done
    - If there exists a node $v$ with degree $< n$
      - Place $v$ on stack to register allocate
    - Else
      - Pick a node $v$ to spill using heuristics (e.g. least frequently executed, with many neighbors etc)
      - Remove $v$ and its edges from the graph
  - If done (no nodes left)
    - Reverse process and add colors
Colorable by 2 Colors?

t1 = 1
t2 = 2
t3 = t1 + t2
t4 = t1 + 3
t5 = t1 + t2

t1-t5 are not live
Colorable by 2 Colors?

Pick t5 and remove it edges

Need to spill!
Is the Algorithm Optimal?

2-colorable? 3-colorable?
Summary

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

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
  - Not successful: colorability unknown and no assignment
backup
Predication Example

\[ a < b \]

\[ s = s + a \]

\[ s = s + b \]

\[ *p = s \]

\[ \text{cmp.lt p1,p2}=a,b \]

\( \begin{align*}
(p1) \quad s &= s + a \\
(p2) \quad s &= s + b \\
* p &= s
\end{align*} \]
Predication

- Identifying region of basic blocks based on resource requirement and profitability (branch misprediction rate, misprediction cost, and parallelism).
- Result: a single predicated basic block
Use the same register for two separate computations in the presence of predication, even if there is live-range overlap without considering predicates.
Example

(p1) \(v1 = 10\)
(p2) \(v2 = 20\) ;;
(p1) \(st4 [v10] = v1\)
(p2) \(v11 = v2 + 1\) ;;

(p1) \(r32 = 10\)
(p2) \(r32 = 20\) ;;
(p1) \(st4 [r33] = r32\)
(p2) \(r34 = r32 + 1\) ;;

overlapped live ranges

same register for \(v1\) and \(v2\)