Register Allocation

Global Register Allocation
Webs and Graph Coloring
Node Splitting and Other Transformations
What a Smart Allocator Needs to Do

• Determine ranges for each variable can benefit from using a register (webs)
• Determine which of these ranges overlap (interference)
• Find the benefit of keeping each web in a register (spill cost)
• Decide which webs get a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)
Global Register Allocation

What’s harder across multiple blocks?
- Could replace a load with a move
- Good assignment would obviate the move
- Must build a control-flow graph to understand inter-block flow
- Can spend an inordinate amount of time adjusting the allocation

This is an assignment problem, not an allocation problem!
Global Register Allocation

A more complex scenario
- Block with multiple predecessors in the control-flow graph
- Must get the “right” values in the “right” registers in each predecessor
- In a loop, a block can be its own predecessors
This adds tremendous complications

What if one block has x in a register, but the other does not?
Outline

• What is Register allocation and Its Importance
• Simple Register Allocators
• Webs
• Interference Graphs
• Graph Coloring
• Splitting
• More Optimizations
Webs

• What needs to Gets Memorized is the Value

• Divide Accesses to a Variable into Multiple Webs
  – All definitions that reaches a use are in the same web
  – All uses that use the value defined are in the same web
  – Divide the Variable into Live Ranges

• Implementation: use DU chains
  – A du-chain connects a definition to all uses reached by the definition
  – A web combines du-chains containing a common use
Example

write y

write x
write y

read x
read y

write x
write x

read x
read y

read x
read x
read x
Example

write y
write x
write y
read y
read x
write x
read x
read x
write y
read y
read x
Example

write y
write x
write y
read y
write x
read x
read x
write x
read x
read x
Example

```
write y  
write x
write y
read x
read x
read y
read x
write x
read x
```
Example

write y
write x
write y
write x
read y
read x
read y
read x
read x
read x
Example

write y

write x
write y

write x
read y

read x
read y

read x
write x

read x
Example

write y

write x
write y

write x
read y

read x
read y

read x
write y

read x
write x

read x
Example

write y

write x
write y

write x
read y

read x
read y

read x
write x

read x
write x
read x
Webs (continued)

• In two Webs of the same Variable:
  – No use in one web will ever use a value defined by the other web
  – Thus, no value need to be carried between webs
  – Each web can be treated independently as values are independent

• Web is used as the Unit of Register Allocation
  – If a web is allocated to a register, all the uses and definitions within that web don’t need to load and store from memory
  – Solves the issue of cross Basic Block register assignment
  – Different webs may be assigned to different registers or one to register and one to memory
Outline

• What is Register Allocation
• A Simple Register Allocator
• Webs
• Interference Graphs
• Graph Coloring
• Splitting
• More Optimizations
Interference

• Two webs interfere if their live ranges overlap in time
  – What does time Mean, more precisely?
  – There exists an instruction common to both ranges where
    • They variable values of webs are operands of the instruction
    • If there is a single instruction in the overlap
      – and the variable for the web that ends at that instruction is an operands and
      – the variable for the web that starts at the instruction is the destination of the instruction
    • then the webs do not interfere

• Non-interfering webs can be assigned to the same register
Example

write y

write x
write y

read x
read y

write x
write y
read x

write x

read x
read y

read x
read y

write x

read x

write x

read x

Example
Example

Webs w1 and w2 interfere
Webs w2 and w3 interfere
Interference Graph

• Representation of Webs & their Interference
  – Nodes are the webs
  – An edge exists between two nodes if they interfere
Example

\[ w_1 \quad w_2 \quad w_3 \quad w_4 \]
Example

Webs w1 and w2 interfere
Webs w2 and w3 interfere

write y
write x
write y
write x
write x
read y
read y
read x
read y
read x
read x
read y
read x

w1
w2
w3
w4
Outline

• What is Register Allocation
• A Simple Register Allocator
• Webs
• Interference Graphs
• Graph Coloring
• Splitting
• More Optimizations
Reg. Allocation Using Graph Coloring

• Each Web is Allocated a Register
  – each node gets a register (color)

• If two webs interfere they cannot use the same register
  – if two nodes have an edge between them, they cannot have the same color
Graph Coloring

• What is the minimum number of colors that takes to color the nodes of the graph such that any nodes connected with an edge does not have the same color?

• Classic Problem in Graph Theory
Graph Coloring Example
Graph Coloring Example

- 1 Color
Graph Coloring Example

Diagram of a graph with nodes and edges.
Graph Coloring Example

- 2 Colors
Graph Coloring Example
Graph Coloring Example

- Still 2 Colors
Graph Coloring Example
Graph Coloring Example

- 3 Colors
Heuristics for Register Coloring

• Coloring a graph with N colors

• If degree < N (degree of a node = # of edges)
  – Node can always be colored
  – After coloring the rest of the nodes, you’ll have at least one color left to color the current node

• If degree ≥ N
  – still may be colorable with N colors
  – exact solution is NP complete
Heuristics for Register Coloring

• Remove nodes that have degree $< N$
  – Push the removed nodes onto a stack

• If all the nodes have degree $\geq N$
  – Find a node to spill (no color for that node)
  – Remove that node

• When empty, start the coloring step
  – Pop a node from stack back
  – Assign it a color that is different from its connected nodes (since degree $< N$, a color should exist)
Coloring Example

$N = 3$
Coloring Example

\[ N = 3 \]
Coloring Example

$N = 3$

![Diagram of a graph with nodes w0, w1, w2, w3, and w4, where w0, w1, and w2 are connected by edges, and w3 is connected to w4.]}
Coloring Example

$N = 3$
Coloring Example

$N = 3$

![Diagram of a coloring example with nodes w0, w1, w2, w3, w4 connected in a way that each node is connected to at most three other nodes, indicating a coloring of the graph.]}
Coloring Example

\[ N = 3 \]
Coloring Example

\[ N = 3 \]

\[ w_0, w_1, w_2, w_3, w_4 \]
Coloring Example

N = 3

\[ \begin{array}{ccc}
  w_0 & w_1 & w_2 \\
  w_3 & w_1 & w_4 \\
\end{array} \]
Coloring Example

$N = 3$

- Blue
- Green
- Brown

Diagram of coloring example with nodes labeled $w_0$, $w_1$, $w_2$, $w_3$, and $w_4$. The nodes are connected in a way that respects the coloring constraints.
Coloring Example

\[ N = 3 \]

\[
\begin{array}{c}
\text{w1} \\
\text{w2} \\
\text{w3} \\
\text{w4} \\
\text{w0}
\end{array}
\]
Coloring Example

$N = 3$

\[ \begin{array}{ccc}
    w1 & w2 & w0 \\
    w3 & w0 & w4 \\
    w2 & w4 & \text{ } \\
\end{array} \]
Coloring Example

$N = 3$

```
N = 3

w0

w1
w2
w3
w4
```
Coloring Example

$N = 3$  

$$w_0 \quad w_1 \quad w_2 \quad w_3 \quad w_4$$
Coloring Example

\[ N = 3 \]

\[
\begin{align*}
  w_1 & \quad w_0 & \quad w_2 \\
  w_3 & \quad w_0 & \quad w_4
\end{align*}
\]
Coloring Example

\[ N = 3 \]

\[ w_0 \]
\[ w_1 \]
\[ w_2 \]
\[ w_3 \]
\[ w_4 \]
Coloring Example

N = 3

w0

w1

w2

w3

w4
Another Coloring Example

\[ N = 3 \]
Another Coloring Example

\[ N = 3 \]
Another Coloring Example

\[ N = 3 \]
Another Coloring Example

\[ N = 3 \]
Another Coloring Example

\[ N = 3 \]
Another Coloring Example

\[ N = 3 \]

\[ s_0 \]
\[ s_1 \]
\[ s_2 \]
\[ s_3 \]
\[ s_4 \]
Another Coloring Example

\[ N = 3 \]

\[ \begin{array}{c}
\text{s0} \\
\text{s1} \\
\text{s2} \\
\text{s3} \\
\text{s4} \\
\end{array} \]
Another Coloring Example

\[ N = 3 \]

\[ \text{s0} \quad \text{s1} \quad \text{s2} \quad \text{s3} \quad \text{s4} \]
Another Coloring Example

$N = 3$
Another Coloring Example

\[ N = 3 \]

\[
\begin{align*}
  &s_0 & s_1 & s_2 \\
  s_0 & & & \\
  s_1 & & & \\
  s_2 & & & \\
  & s_3 & s_4 \\
\end{align*}
\]
Another Coloring Example

\[ N = 3 \]

\[
\begin{array}{c}
s0 \\
\hline
s1 \\
\hline
s2 \\
\hline
s3 \\
\hline
s4
\end{array}
\]
Another Coloring Example

\[ N = 3 \]
Another Coloring Example

\[ N = 3 \]

![Graph with nodes s0, s1, s2, s3, s4 colored with three colors](image-url)
Outline

• What is Register Allocation
• A simple register Allocator
• Webs
• Interference Graphs
• Graph coloring
• Splitting
• More Optimizations
Spilling and Splitting

• When the graph is non-N-colorable

• Select a Web to Spill
  – Find the least costly Web to Spill
  – Use and Defs of that web are read and writes to memory

• Split the web
  – Split a web into multiple webs so that there will be less interference in the interference graph making it N-colorable
  – Spill the value to memory and load it back at the points where the web is split
Splitting Example

write z
read z
write x
write y
read x
read x
read y
read z
Splitting Example

write z
read z

write x
write y
read x
read x
read y

read z

x y z
Splitting Example

write z
read z

write x
write y
read x
read x
read y

read z

2 colorable?
Splitting Example

write z
read z

write x
write y
read x
read x
read y
read z

2 colorable? NO!
Splitting Example

write z
read z

write x
write y
read x
read x
read y

read z
Splitting Example

write z
read z

write x
write y
read x
read x
read y

read z
Splitting Example

write z
read z

write x
write y
read x
read x
read y

read z

x
y
z1
z2
Splitting Example

write z
read z
write x
write y
read x
read x
read y
read z

2 colorable?
Splitting Example

write z
read z
write x
write y
read x
read x
read y
read z

2 colorable? YES!
Splitting Example

write z
read z

write x
write y
read x
read x
read y

write z

2 colorable? YES!
Splitting Example

write z
read z
store z

write x
write y
read x
read y

load z
read z

2 colorable?
YES!
Splitting

• Identify a Program Point where the Graph is not R-colorable (point where # of webs > N)
  – Pick a web that is not used for the largest enclosing block around that point of the program
  – Split that web
  – Redo the interference graph
  – Try to re-color the graph
Cost and Benefit of Splitting

• Cost of splitting a node
  – Proportion to number of times splitted edge has to be crossed dynamically
  – Estimate by its loop nesting

• Benefit
  – Increase colorability of the nodes the splitted web interferes with
  – Can approximate by its degree in the interference graph

• Greedy heuristic
  – pick the live-range with the highest benefit-to-cost ratio to spill
Outline

- Overview of procedure optimizations
- What is register allocation
- A simple register allocator
- Webs
- Interference Graphs
- Graph coloring
- Splitting
- More Optimizations
More Transformations

• Register Coalescing
• Register Targeting (pre-coloring)
• Pre-Splitting of Webs
• Inter-procedural Register Allocation
Register Coalescing

- Find register copy instructions \( s_j = s_i \)
- If \( s_j \) and \( s_i \) do not interfere, combine their webs
- Pros
  - Similar to copy propagation
  - Reduce the number of instructions
- Cons
  - May increase the degree of the combined node
  - A colorable graph may become non-colorable
Register Targeting (pre-coloring)

- Some Variables need to be in Special Registers at Specific Points in the Execution
  - first 4 arguments to a function
  - return value

- Pre-color those webs and bind them to the appropriate register

- Will eliminate unnecessary copy instructions
Pre-Splitting of the Webs

• Some Ranges have Very Large “dead” Regions
  – Large region where the variable is unused

• Break-up the Ranges
  – need to pay a small cost in spilling
  – but the graph will be very easy to color

• Can find Strategic Locations to Break-up
  – at a call site (need to spill anyway)
  – around a large loop nest (reserve registers for values used in the loop)
Inter-Procedural Register Allocation

• Saving Registers across Procedure boundaries is expensive
  – especially for programs with many small functions
• Calling convention is too general and inefficient
• Customize calling convention per function by doing inter-procedural register allocation
Chaitin-Briggs Allocator

1. **renumber**
   - Build SSA, build live ranges, rename

2. **build**
   - Build the interference graph

3. **coalesce**
   - Fold unneeded copies
     - $LR_x \rightarrow LR_y$, and $<LR_x, LR_y> \notin G_i \Rightarrow$ combine $LR_x$ & $LR_y$

4. **spill costs**
   - Estimate cost for spilling each live range

5. **simplify**
   - Remove nodes from the graph

6. **select**
   - While stack is non-empty
     - pop $n$, insert $n$ into $G_i$, & try to color it

7. **spill**
   - Spill uncolored definitions & uses

Briggs’ algorithm (1989)
Summary

• Register Allocation and Assignment
  – Very Important Transformations and Optimization
  – In General Hard Problem (NP-Complete)

• Many Approaches
  – Local Methods: Top-Down and Bottom-Up
  – Global Methods: Graph Coloring
    • Webs
    • Interference Graphs
    • Coloring
  – Other Transformations