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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)

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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)

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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)

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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)

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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)

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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)

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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)

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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)

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 gets a register (allocation)
• Split webs if needed (spilling and splitting)
• Assign hard registers to webs (assignment)
• Generate code including spills (code gen)
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

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 operand 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

Webs $s_1$ and $s_2$ interfere
Webs $s_2$ and $s_3$ interfere

Interference Graph

- Representation of webs and their interference
  - Nodes are the webs
  - An edge exists between two nodes if they interfere

Webs $s_1$ and $s_2$ interfere
Webs $s_2$ and $s_3$ interfere

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

- 1 Color

Graph Coloring Example

- 2 Colors
Graph Coloring Example

Still 2 Colors

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 ≥ 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

\( N = 3 \)

\[ \begin{align*}
s_0 & \quad s_1 \\
| & \quad | \\
\text{Red} & \quad \text{Blue} \\
\end{align*} \]
Coloring Example

N = 3

s0
s1
s2
s3
s4
Coloring Example

N = 3

Another Coloring Example

N = 3
Another Coloring Example

$N = 3$

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

N = 3

s0
s1
s2
s3
s4

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

def z
  use z

def x
  use x
  use x
  use y

use z

x
  y
  z

2 colorable?

NO!
Splitting Example

```python
def z
  use z

def x
def y
  use x
  use y

use 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 split 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 live ranges have very large “dead” regions.
  - Large region where the variable is unused

- Break-up the live 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

- Build SSA, build live ranges, rename
- Build the interference graph
- Fold unneeded copies
  \[ LR_x \rightarrow LR_y \text{ and } LR_x \cup LR_y \subseteq G \Rightarrow \text{combine } LR_x \text{ & } LR_y \]
- Estimate cost for spilling each live range
- Remove nodes from the graph
- While stack is non-empty
  - Pop \( n \) from stack
  - Insert \( n \) into \( G \)
  - Try to color it
  - Spill uncolored definitions & uses

Briggs’ algorithm (1989)