

Register Allocation

CS 471
November 12, 2007



Register Allocation - Motivation

Consider adding two numbers together:

Option #1: Keep Values in Memory	Option #2: Keep Values in Registers
load r1, 4(sp)	add r3,r1,r2
load r2, 8(sp)	
add r3,r1,r2	
store r3, 12(sp)	

Advantages:
Fewer instructions – most instrs are reg ↔ reg
Cheaper instructions – accessing memory is expensive

CS 471 – Fall 2007

Register Allocation – The Catch

Very few registers available!!
Some registers must be used for a particular purpose

- Integer vs. floating-point registers
- Doubleword registers (odd/even pairs)
- Branch and predicate registers
- Stack pointer
- RO
- String manipulation (implicit uses)

CS 471 – Fall 2007

Register Allocation

Register allocation – determines which of the values (variables, temporaries, large constants) should be in registers at each execution point

Register assignment – determines which register should be used by each allocated value

Goal - Want to minimize the traffic between the CPU registers and the memory hierarchy

Probably the optimization with the most performance impact!

CS 471 – Fall 2007

A Simple Example

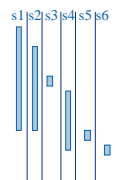
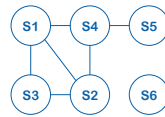
Original code	Symbolic register assignment
x ← 2	s1 ← 2
y ← 4	s2 ← 4
w ← x + y	s3 ← s1 + s2
z ← x + 1	s4 ← s1 + 1
u ← x * y	s5 ← s1 * s2
x ← z * 2	s6 ← s4 * 2

Gen – variable is newly defined **Def** – on LHS
Kill – old value no longer accessible **Use** – on RHS

SSA – Static single assignment

CS 471 – Fall 2007

A Simple Example

s1 ← 2		
s2 ← 4		
s3 ← s1 + s2		
s4 ← s1 + 1		
s5 ← s1 * s2		
s6 ← s4 * 2		

Live – variable will be used again (before it is overwritten)
Dead

- Variable will never be used again
- Value will be overwritten before next use

CS 471 – Fall 2007

A Simple Example

```

s1 ← 2
s2 ← 4
s3 ← s1 + s2
s4 ← s1 + 1
s5 ← s1 * s2
s6 ← s4 * 2

```

Graph Coloring

r1 ← green
r2 ← blue
r3 ← pink

Result

```

r1 ← 2
r2 ← 4
r3 ← r1 + r2
r3 ← r1 + 1
r1 ← r1 * r2
r2 ← r3 * 2

```

CS 471 – Fall 2007

Register Allocation

- Determine live ranges for each value (web)
- Determine overlapping ranges (interference)
- Compute 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)

CS 471 – Fall 2007

Webs

Starting Point: def-use chains (DU chains)

- Connects definition to all reachable uses

Conditions for putting defs and uses into same web

- Def and all reachable uses must be in same web
- All defs that reach same use must be in same web

CS 471 – Fall 2007

Example

CS 471 – Fall 2007

Interference

- Two webs **interfere** if their live ranges overlap (have a nonempty intersection)
- If two webs interfere, values must be stored in different registers or memory locations
- If two webs do not interfere, can store values in same register or memory location

CS 471 – Fall 2007

Example

Webs s1 and s2 interfere
Webs s2 and s3 interfere

CS 471 – Fall 2007

Interference Graph

Representation of webs and their interference

- Nodes are the webs
- An edge exists between two nodes if they interfere

CS 471 – Fall 2007

Example

Webs s1 and s2 interfere
Webs s2 and s3 interfere

CS 471 – Fall 2007

Register 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

CS 471 – Fall 2007

Graph Coloring

Assign a color to each node in graph

Two nodes connected to same edge must have different colors

Classic problem in graph theory

NP complete

- But good heuristics exist for register allocation

CS 471 – Fall 2007

Graph Coloring Example #1

CS 471 – Fall 2007

Graph Coloring Example #2

CS 471 – Fall 2007

Graph Coloring Example #3

CS 471 – Fall 2007

Graph Coloring Example #4

CS 471 – Fall 2007

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 $\geq N$

- still may be colorable with N colors

CS 471 – Fall 2007

Heuristics for Register Coloring

Remove nodes that have degree $< N$

- Push the removed nodes onto a stack

When all the nodes have degree $\geq N$

- Find a node to spill (no color for that node)
- Remove that node

When empty, start to color

- Pop a node from stack back
- Assign it a color that is different from its connected nodes (since degree $< N$, a color should exist)

CS 471 – Fall 2007

Coloring Example #5

$N = 3$

CS 471 – Fall 2007

Coloring Example #5

$N = 3$

CS 471 – Fall 2007

Coloring Example #5

$N = 3$ [blue square] [green square] [brown square]

s0
s3
s1
s2
s4

CS 471 - Fall 2007

Coloring Example #5

$N = 3$ [blue square] [green square] [brown square]

s3
s1
s2
s4

CS 471 - Fall 2007

Coloring Example #5

$N = 3$ [blue square] [green square] [brown square]

s1
s2
s4

CS 471 - Fall 2007

Coloring Example #5

$N = 3$ [blue square] [green square] [brown square]

s2
s4

CS 471 - Fall 2007

Coloring Example #5

$N = 3$ [blue square] [green square] [brown square]

s4

CS 471 - Fall 2007

Coloring Example #5

$N = 3$ [blue square] [green square] [brown square]

CS 471 - Fall 2007

Another Coloring Example

$N = 3$

CS 471 – Fall 2007

Another Coloring Example

$N = 3$

s4

CS 471 – Fall 2007

Another Coloring Example

$N = 3$

s4

CS 471 – Fall 2007

Another Coloring Example

$N = 3$

s3
s4

CS 471 – Fall 2007

Another Coloring Example

$N = 3$ ■ ■ ■

s0
s2
s3
s4

CS 471 – Fall 2007

Another Coloring Example

$N = 3$ ■ ■ ■

s2
s3
s4

CS 471 – Fall 2007

Another Coloring Example

$N = 3$

CS 471 – Fall 2007

When Coloring Fails?

Option 1

- Pick a web and allocate value in memory
- All defs go to memory, all uses come from memory

Option 2

- Split the web into multiple webs

In either case, will retry the coloring

CS 471 – Fall 2007

Which web to choose?

One with interference degree $\geq N$
 One with minimal **spill cost** (cost of placing value in memory rather than in register)

What is spill cost?

- Cost of extra load and store instructions

CS 471 – Fall 2007

Ideal and Useful Spill Costs

Ideal spill cost - dynamic cost of extra load and store instructions. Can't expect to compute this.

- Don't know which way branches resolve
- Don't know how many times loops execute
- Actual cost may be different for different executions

Solution: Use a static approximation

- profiling can give instruction execution frequencies
- or use heuristics based on structure of control flow graph

CS 471 – Fall 2007

One Way to Compute Spill Cost

Goal: give priority to values used in loops
 So assume loops execute 8 or 10 times

Spill cost =

- sum over all def sites: cost of a store instruction times 10 to the loop nesting depth power, plus
- sum over all use sites: cost of a load instruction times 10 to the loop nesting depth power

Choose the web with the lowest spill cost

CS 471 – Fall 2007

Spill Cost Example

Spill Cost For x
 $\text{storeCost} + \text{loadCost}$

Spill Cost For y
 $10 * \text{storeCost} + 10 * \text{loadCost}$

With 1 Register, Which Variable Gets Spilled?

CS 471 – Fall 2007

Outline

- What is register allocation
- A simple register allocator
- Webs
- Interference Graphs
- Graph coloring
- Splitting
- More optimizations

CS 471 – Fall 2007

Splitting Rather Than Spilling

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

CS 471 – Fall 2007

Splitting Example

CS 471 – Fall 2007

Splitting Example

CS 471 – Fall 2007

Splitting Example

2 colorable?

CS 471 – Fall 2007

Splitting Example

CS 471 – Fall 2007

Splitting Example

2 colorable?

CS 471 – Fall 2007

Splitting Example

2 colorable?
YES!

CS 471 – Fall 2007

Splitting Example

2 colorable?
YES!

CS 471 – Fall 2007

Splitting Heuristic

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 at the corresponding edge
- Redo the interference graph
- Try to re-color the graph

CS 471 – Fall 2007

Cost and benefit of splitting

Cost of splitting a node

- Proportional to number of times split edge has to be crossed dynamically
- Estimate by its loop nesting

Benefit

- Increase colorability of the nodes the split 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

CS 471 – Fall 2007

Further Optimizations

- Register coalescing
- Register targeting (pre-coloring)
- Presplitting of webs
- Interprocedural register allocation

CS 471 – Fall 2007

Register Coalescing

Find register copy instructions $s_j = s_i$
If s_j and s_i do not interfere, combine their webs

Pros

- reduce the number of instructions

Cons

- may increase the degree of the combined node
- a colorable graph may become non-colorable

CS 471 – Fall 2007

Register Targeting (pre-coloring)

Some variables need to be in special registers at a given time

- first 4 arguments to a function
- return value

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

Will eliminate unnecessary copy instructions

CS 471 – Fall 2007

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)

CS 471 – Fall 2007

Interprocedural register allocation

Saving/restoring 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 interprocedural register allocation

CS 471 – Fall 2007

Summary

Register Allocation

- Store values in registers between def and use
- Can improve performance substantially

Key concepts

- Webs
- Interference graphs
- Colorability
- Splitting

CS 471 – Fall 2007