KT-2

P3 / 2006

Register Allocation

Outline

- What is register allocation
- Webs
- Interference Graphs
- Graph Coloring
- Spilling
- Live-Range Splitting
- More Variations and Optimizations

Storing values between defs and uses

- Program computes with values
  - value definitions (where computed)
  - value uses (where read to compute new values)
- Values must be stored between def and use

  First Option:
  - store each value in memory at definition
  - retrieve from memory at each use

  Second Option:
  - store each value in register at definition
  - retrieve value from register at each use

Issues

- On a typical RISC architecture
  - All computation takes place in registers
  - Load instructions and store instructions transfer values between memory and registers
- Add two numbers; values in memory

```
load r1, 4(sp)
load r2, 8(sp)
add r3,r1,r2
store r3, 12(sp)
```


Issues

- On a typical RISC architecture
  - All computation takes place in registers
  - Load instructions and store instructions transfer values between memory and registers
- Add two numbers; values in registers

```
add r3,r1,r2
```


Issues

- Fewer instructions when using registers
  - Most instructions are register-to-register
  - Additional instructions for memory accesses
- Registers are faster than memory
  - Wider gap in faster, newer processors
  - Factor of about 4 bandwidth, factor of about 3 latency
  - Could be bigger depending on program characteristics
- But only a small number of registers available
  - Usually 32 integer and 32 floating-point registers (e.g. on SPARC, MIPS, or PowerPC), or much less on x86 or AMD64
  - Some of those registers have fixed users (r0, ra, sp, fp)

Register Allocation

- Deciding which values to store in a limited number of registers
- Register allocation has a direct impact on performance
  - Affects almost every statement of the program
  - Eliminates expensive memory instructions
  - # of instructions goes down due to direct manipulation of registers (no need for load and store instructions)
  - Probably the optimization with the most impact!

What can be put in a register?

- Values stored in compiler-generated temps
- Language-level values
  - Values stored in local scalar variables
  - Big constants
  - Values stored in array elements and object fields
    - Issue: alias analysis
- Register set depends on the data-type
  - floating point values in floating point registers
  - integer and pointer values in integer registers

Outline

- What is Register Allocation
- Webs
- Interference Graphs
- Graph Coloring
- Spilling
- Live-Range Splitting
- More optimizations

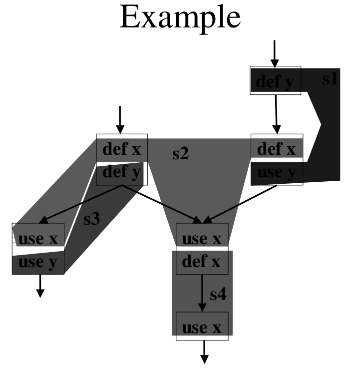
Web-Based Register Allocation

- 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 generation)

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
- Use a union-find algorithm

Example



Webs

- Web is unit of register allocation
- If web allocated to a given register R
  - All definitions computed into R
  - All uses read from R
- If web allocated to a memory location M
  - All definitions computed into M
  - All uses read from M
- Issue: instructions compute only from registers
- Reserve some registers to hold memory values

Outline

- What is Register Allocation
- Webs
- Interference Graphs
- Graph Coloring
- Spilling
- Live-Range Splitting
- More Variations and Optimizations

Convex Sets and Live Ranges

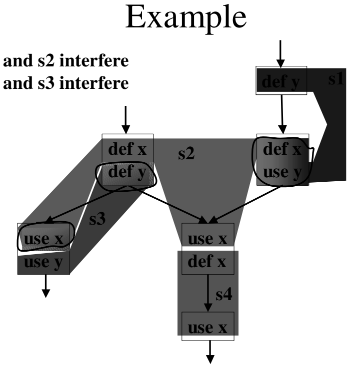
- Concept of convex set
- A set S is convex if
  - A, B in S and C is on a path from A to B implies
  - C is in S
- Concept of live range of a web
  - Minimal convex set of instructions that includes all defs and uses in web
  - Intuitively, region in which web's value is live

Interference

- Two webs interfere if their live ranges overlap (have a non-empty 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

Example

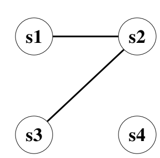
Webs s1 and s2 interfere
Webs s2 and s3 interfere



Interference Graph

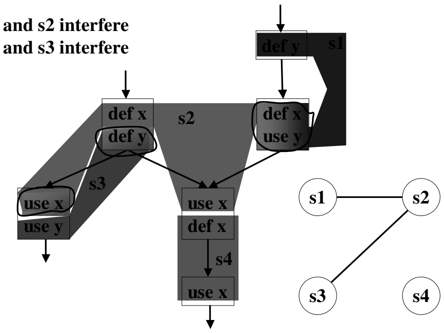
Representation of webs and their interference

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



Example

Webs s1 and s2 interfere
Webs s2 and s3 interfere

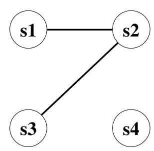


Outline

- What is Register Allocation
- Webs
- Interference Graphs
- Graph Coloring
- Spilling
- Live-Range Splitting
- More optimizations

Register Allocation Using Graph Coloring

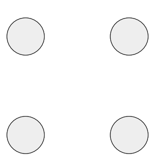
- Each web is allocated a register
  - each node gets a register (color)
- If two webs interfere they cannot use the same register
  - if two nodes have an edge between them, they cannot have the same color



Graph Coloring

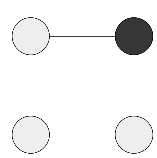
- Assign a color to each node in graph
- Two nodes connected to same edge must have different colors
- Classic problem in graph theory
- NP complete
  - But good heuristics exist for register allocation

Graph Coloring Example



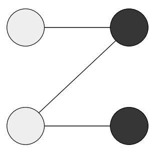
1 Color

Graph Coloring Example



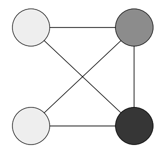
2 Colors

Graph Coloring Example



Still 2 Colors

Graph Coloring Example



3 Colors

Heuristics for Register Coloring

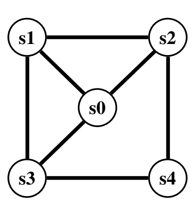
- Coloring a graph with N colors
- If degree < N (degree of a node = # of edges)
  - Node can always be colored
  - After coloring the rest of the nodes, there is at least one color left to color the current node
- If degree >= N
  - still may be colorable with N colors

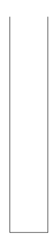
Heuristics for Register Coloring

- Remove nodes that have degree < N
  - Push the removed nodes onto a stack
- When all the nodes have degree >= N
  - Find a node to spill (no color for that node)
  - Push that node into the stack
- When graph becomes empty, start to color
  - Pop a node from stack back
  - Assign it a color that is different from its connected nodes (if possible)

Coloring Example

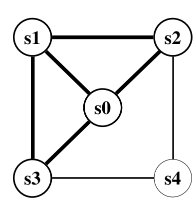
N = 3

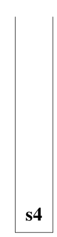




Coloring Example

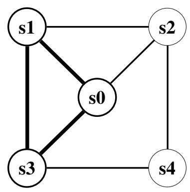
N = 3





Coloring Example

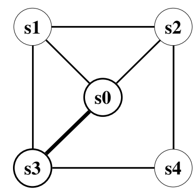
N = 3



s2
s4

Coloring Example

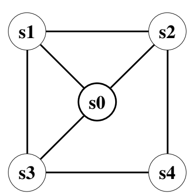
N = 3



s1
s2
s4

Coloring Example

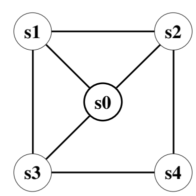
N = 3



s3
s1
s2
s4

Coloring Example

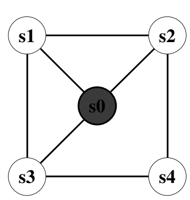
N = 3



s3
s1
s2
s4

Coloring Example

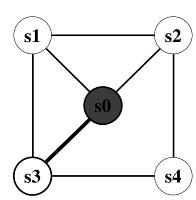
N = 3



s3
s1
s2
s4

Coloring Example

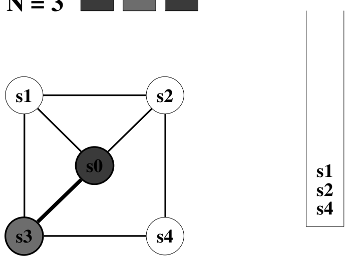
N = 3



s1
s2
s4

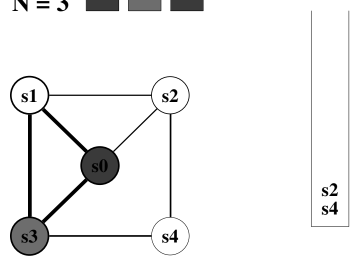
Coloring Example

N = 3



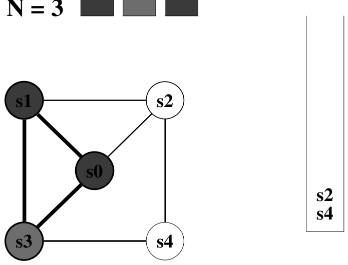
Coloring Example

N = 3



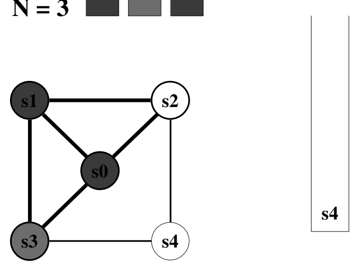
Coloring Example

N = 3



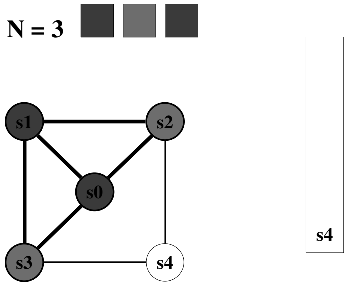
Coloring Example

N = 3



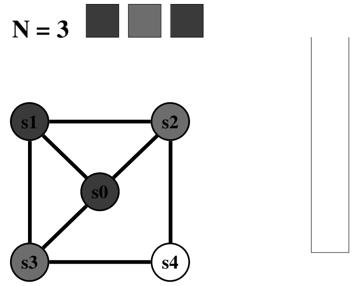
Coloring Example

N = 3



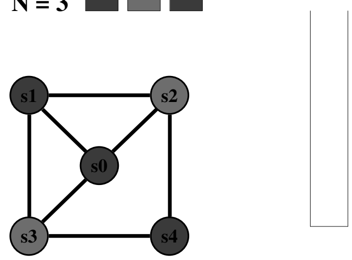
Coloring Example

N = 3



Coloring Example

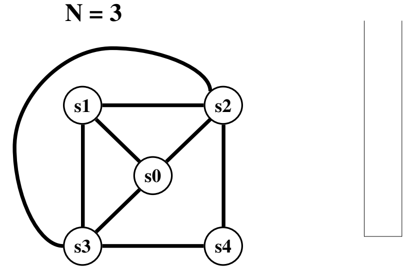
N = 3



Kostis Sagonas 43 Spring 2006

Another Coloring Example

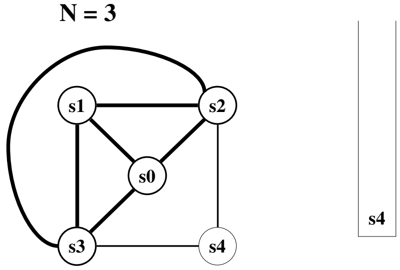
N = 3



Kostis Sagonas 44 Spring 2006

Another Coloring Example

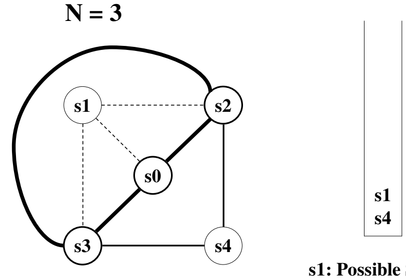
N = 3



Kostis Sagonas 45 Spring 2006

Another Coloring Example

N = 3

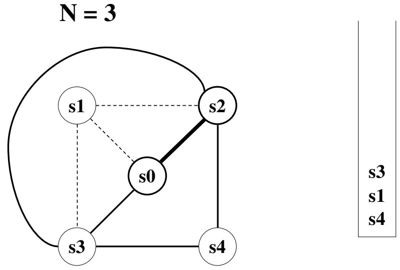


s1: Possible Spill

Kostis Sagonas 46 Spring 2006

Another Coloring Example

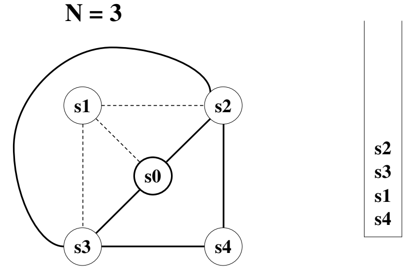
N = 3



Kostis Sagonas 47 Spring 2006

Another Coloring Example

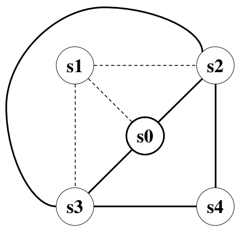
N = 3



Kostis Sagonas 48 Spring 2006

Another Coloring Example

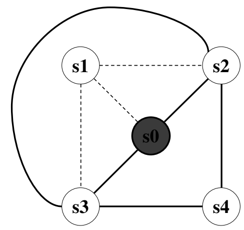
N = 3



s2
s3
s1
s4

Another Coloring Example

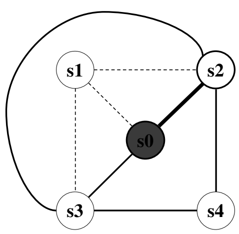
N = 3



s2
s3
s1
s4

Another Coloring Example

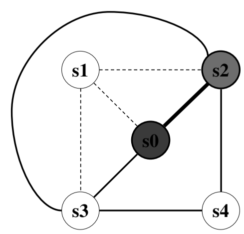
N = 3



s3
s1
s4

Another Coloring Example

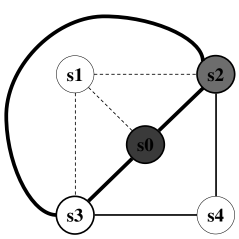
N = 3



s3
s1
s4

Another Coloring Example

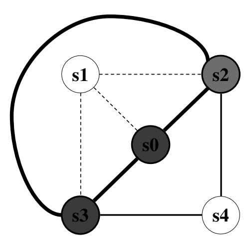
N = 3



s1
s4

Another Coloring Example

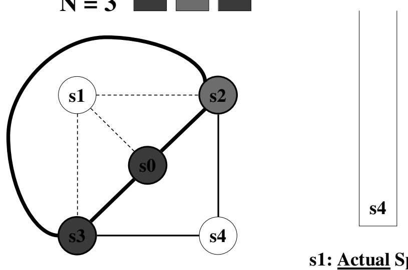
N = 3



s1
s4

Another Coloring Example

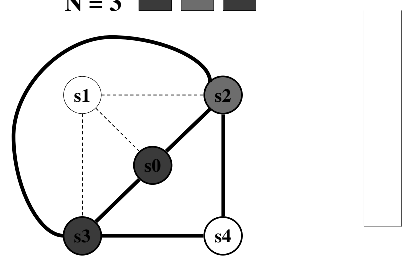
N = 3



s1: Actual Spill

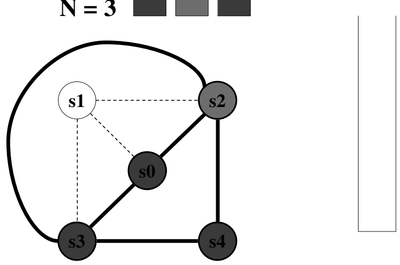
Another Coloring Example

N = 3



Another Coloring Example

N = 3



When Coloring Heuristics Fail...

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

Outline

- What is Register Allocation
- Webs
- Interference Graphs
- Graph Coloring
- Spilling
- Live-Range Splitting
- More Variations and Optimizations

Which web to pick?

- One with interference degree >= 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

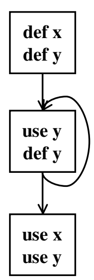
Ideal and Useful Spill Costs

- 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

One Way to Estimate Spill Cost

- Goal: give priority to values used in loops
- So assume loops execute 10 (or 8) times
- Estimated spill cost =
  - sum over all def sites of cost of a store instruction times 8 to the loop nesting depth power, plus
  - sum over all use sites of cost of a load instruction times 8 to the loop nesting depth power
- Choose the web with the lowest spill cost

Spill Cost Example



Spill Cost For x
storeCost+loadCost

Spill Cost For y
9*storeCost+9*loadCost

With 1 Register, Which Variable Gets Spilled?

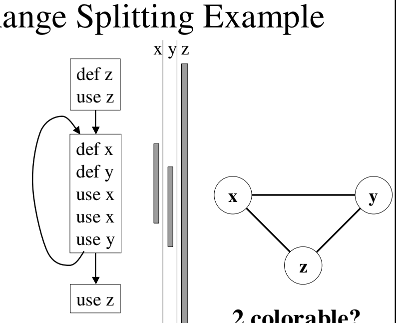
Outline

- What is Register Allocation
- Webs
- Interference Graphs
- Graph Coloring
- Spilling
- Live-Range Splitting
- More Variations and Optimizations

Splitting Rather Than Spilling

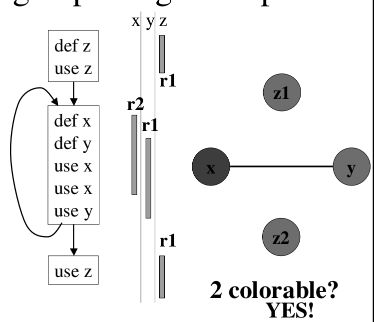
- Split the web
  - Split a web into multiple webs so that there will be less interference in the interference graph making it N-colorable
  - Spill the value to memory and load it back at the points where the web is split

Live-Range Splitting Example

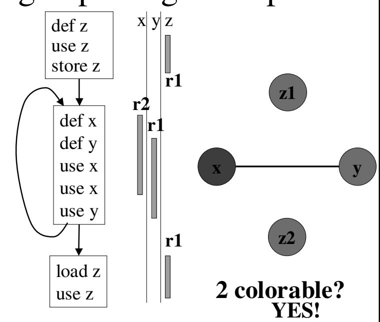


2 colorable?
NO!

Live-Range Splitting Example



Live-Range Splitting Example



Live-Range Splitting Heuristic

- Identify a program point where the graph is not N-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

Cost and Benefit of Splitting

- Cost of splitting a node
  – Proportional to number of times split edge has to be crossed dynamically
  – Estimate this number by its loop nesting
- Benefit
  – Increases 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

Outline

- What is Register Allocation
- Webs
- Interference Graphs
- Graph Coloring
- Splitting
- Live-Range Splitting
- More Variations and Optimizations

Further Optimizations

- Register coalescing
  – Briggs-style coalescing
  – Iterated register coalescing
  – Optimistic register coalescing
- Register targeting (pre-coloring)
- Pre-splitting of webs
- Interprocedural 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
  - reduces the number of instructions
- Cons
  - may increase the degree of the combined node
  - a colorable graph may become non-colorable
    - however, safe coalescing criteria exist

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 right 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 might become very easy to color
- Can find strategic locations to break-up
  - at a call site (may need to spill anyway)
  - around a large loop nest (reserve registers for values used in the loop)

Interprocedural 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 interprocedural register allocation