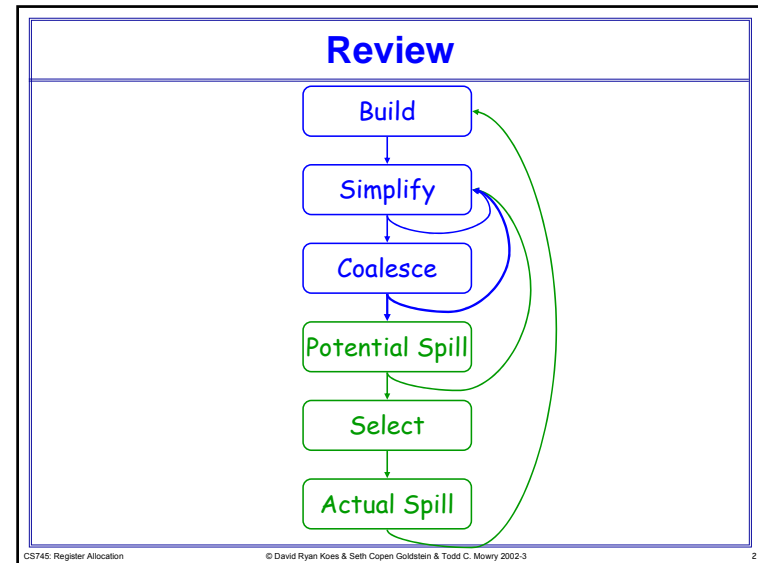


15-745 Register Allocation Spilling & Stuff

STUPID SOFTWARE!
WON'T COMPILE;
EH??

WE CALL IT "CODE RAGE." I'M SEEING A LOT OF IT LATELY.

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 1



Review: Build

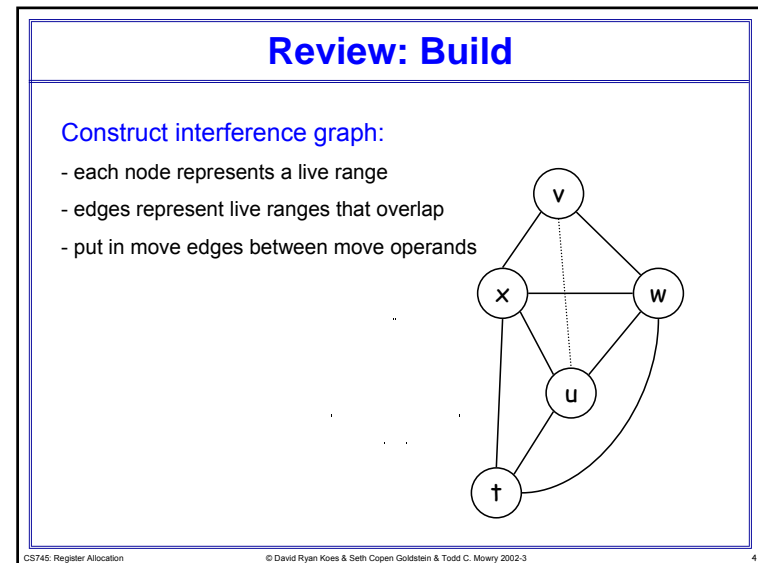
```

v <- 1
w <- v + 3
x <- w + v
u <- v
t <- u + x
<- w
<- t
<- u
  
```

First compute live ranges:

- use both reach defs and liveness
- live range defined by definition point
- ends when variable dies
- merge overlapping ranges of same var

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 3



Review: Simplify

Reduce the graph:

- remove non-move related, easy to color, nodes
- easy to color: degree < k
- place on stack

$k = 4$

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 5

Review: Coalesce

Coalesce moves:

- conservatively combine operands of a move
- Briggs, George heuristics for being conservative

Repeat Simplify

-Detail: If both Simplify and Coalesce get stuck, start simplifying move related nodes

$k = 4$

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 6

Transition Slide!

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 7

What if we can't simplify?

Now what?

Be optimistic:

- Put a node with degree $\geq k$ on stack
- Lose guarantee that anything we put on stack is colorable
- If we're lucky this node will still be colorable when popped from stack

Be realistic:

- If unlucky, this node will have to be spilled (allocated to memory)
- Mark as *potential spill* to avoid recomputation later

$k = 3$

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 8

Select

Pop a node from the stack

Assign it a color that does not conflict with neighbors in interference graph

This will always be possible, unless the node is a **potential spill**

If it is not possible **must spill**

$k = 3$

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 9

Spilling to Memory

RISC Architectures

- Only load and store can access memory
 - every use requires load
 - every def requires store
 - create new temporary for each location

CISC Architectures

- can operate on data in memory directly
 - makes writing compiler easier(?), but isn't necessarily faster
- pseudo-registers inside memory operands still have to be handled

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 10

Spilling

```

v <- 1
w1 <- v + 3
Mw[] <- w1
w2 <- Mw[]
x <- w2 + v
u <- v
t <- u + x
w3 <- Mw[]
<- w3
<- t
<- u
    
```

Allocate w to memory location M_w

Spilled variables are allocated to the stack in an area completely controlled by the compiler. These memory locations are special in that they can be optimized without concern for memory aliasing issues.

Now Start Over...
...compute live ranges...

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 11

Build Take Two

```

v <- 1
w1 <- v + 3
Mw[] <- w1
w2 <- Mw[]
x <- w2 + v
u <- v
t <- u + x
w3 <- Mw[]
<- w3
<- t
<- u
    
```

$k = 3$

Recalculate interference graph

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 12

Simplify->Coalesce->Select

$k = 3$

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 13

Spilling

We have to start from scratch every time we spill

- **Suggestions?**
 - Fewer iterations?
 - Faster iterations?

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 14

What to Spill?

When choosing potential spill node want:

- **A node that makes graph easier to color**
 - Fewer spills later
- **A node that isn't "expensive" to spill**
 - First nodes pushed on stack are last to be colored
 - more likely to be spilled
 - An expensive node would slow down the program if spilled
- **We can apply heuristics both when choosing potential spill nodes and when choosing actual spill nodes**
 - not required to spill node that we popped off stack and can't color

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 15

A Spill Heuristic

Pick node (live range) n that minimizes:

$$\frac{\sum_{def \in n} 10^{depth(def)} + \sum_{use \in n} 10^{depth(use)}}{degree(n)}$$

This heuristic prefers nodes that:

- Are used infrequently
- Aren't used inside of loops
- Have a large degree

Could use any one of several other heuristics as well...

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 16

Reducing Stack Frame Size

- How do you allocate spilled live ranges?
 - every live range gets its own location on the stack frame
 - or we can be smarter...
- What about `mov a, b` where both a & b have been spilled?
- Use graph-coloring with aggressive coalescing!
- Use liveness info to create an interference graph of the spilled nodes
- Always coalesce
- Simplify/Select
- Colors map to frame locations

Is it worth it?

CS745: Register Allocation


© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

17

Rematerialization

An alternative to spilling

- **Recompute value of variable instead of store/load to memory**
- **Example:**

<code>v <- 1</code>		<code>v <- 1</code>
<code>w <- v + 3</code>		<code>w <- v + 3</code>
<code>x <- w + v</code>		<code>x <- w + v</code>
<code>u <- v</code>		<code>u <- v</code>
<code>t <- u + x</code>		<code>t <- u + x</code>
<code><- w</code>		<code>w <- 4</code>
<code><- t</code>		<code><- w</code>
<code><- u</code>		<code><- t</code>
		<code><- u</code>

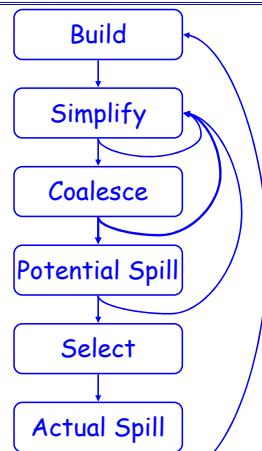
CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

18

Checkpoint

Talk about projects



CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

19

Special Registers

Which registers can be used?

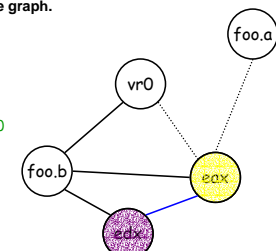
- Some registers have **special uses**.
 - Register 0 or 31 is often hardwired to contain 0.
 - Special registers to hold return address, stack pointer, frame pointer, etc.
 - Reserved registers for operating system.
- Typically, leaves about 20 or so registers for other general uses.

Impact on register allocation:

- Temps should be assigned only to the non-reserved registers (allocable).
- Hard registers are pre-colored in the interference graph.

```

movl    foo.a,%eax
cld    (eax,edx) <- eax
idivl  foo.b (eax,edx) <- (eax,edx)/foo.b
movl    %eax,$vr0
movl    $vr0,%eax
ret
    
```



CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

20

Register Usage Conventions

Certain registers are used for specific purposes defined by the *standard calling convention*.

- **4-6 argument registers.**
 - The first 4-6 arguments to procedures/functions are always passed in these registers.
- **~8 callee-save registers.**
 - These registers **must be preserved across procedure calls**. Thus, if a procedure wants to use a callee-save register, it must first save the old value and then restore it before returning.
- **The remainder are caller-save registers.**
 - These are **not preserved across procedure calls**. Thus, a procedure is free to use them without saving first.
 - Includes the argument registers.

How do we support these?

- neat trick for handling callee save
- call instruction

CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

21

Allocating Callee-Save Registers

Move callee-save reg to temp at start of procedure
 Move it back at end of procedure
 What happens if there is no register pressure?
 What happens if there is a lot of register pressure?

```
entry: define r
    temp r
    ...
exit: r temp
    use r
```

CS745: Register Allocation

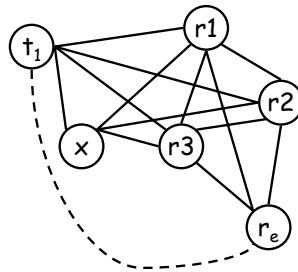
© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

22

Allocating to callee-save registers

CALL instruction “defines” all caller-save regs

```
entry: define re
    t1 ← re
    x ←
    ...
    call
    ...
    ← x
    ← x
exit: re ← t1
    use re
```



CS745: Register Allocation

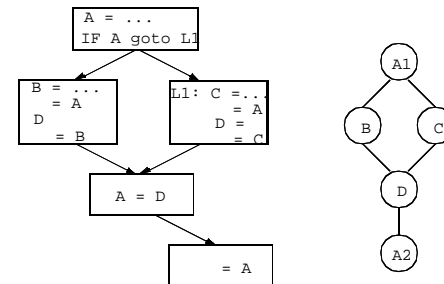
© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

23

Reducing Register Pressure

Recall: Split pseudo-registers into live ranges to create an interference graph that is easier to color

- Eliminate interference in a variable's “dead” zones.
- Increase flexibility in allocation: can allocate same variable to different registers



CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

24

Insight

Split a live range into smaller regions (by paying a small cost) to create an interference graph that is easier to color

- Eliminate interference in a variable's "nearly dead" zones.
 - Cost: Memory loads and stores
Load and store at boundaries of regions with no activity
 - # active live ranges at a program point can be > # registers
- Can allocate same variable to different registers
 - Cost: Register operations
a register copy between regions of different assignments
 - # active live ranges cannot be > # registers

CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

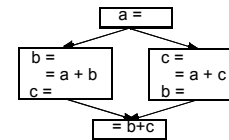
25

Examples

Example 1:

```
FOR i = 0 TO 10
  FOR j = 0 TO 10000
    A = A + ...
    (does not use B)
  FOR j = 0 TO 10000
    B = B + ...
    (does not use A)
```

Example 2:



CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

26

Live-Range Splitting

When do we apply live range splitting?

Which live range to split?

Where should the live range be split?

How to apply live-range splitting with coloring?

- Advantage of coloring:
 - defers arbitrary assignment decisions until later
- When coloring fails to proceed, may not need to split live range
 - degree of a node $\geq n$ does not mean that the graph definitely is not colorable
- Interference graph does not capture positions of a live range

CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

27

One Algorithm

Observation: Spilling is absolutely necessary if

- number of live ranges active at a program point > n *not degree in graph*

Apply live-range splitting before coloring

- Identify a point where number of live ranges > n
- For each live range active around that point
 - find the outermost "block construct" that does not access the variable
- Choose a live range with the largest inactive region
- Split the inactive region from the live range

CS745: Register Allocation

© David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3

28

Alternative Allocators

Graph allocator, as described, has issues

- What are they?

Alternative: Single pass graph coloring

- **Build, Simplify, Coalesce as before**
- **In select, if can't color with register, color with stack location**
 - Keep going
- **Requires second, reload phase**
 - "fixes" spilled variables
 - Requires that we reserve a register
 - Can get messy

Claim: Does a pretty good job

- **Why?**
 - Key is order nodes are colored...

Advantages? Disadvantages?

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 29

Alternative Allocators

Local/Global Allocation

- **Allocate "local" pseudo-registers**
 - Lifetime contained within basic block
 - No longer NP-Complete!
- **Allocate global pseudo-registers**
 - Single pass global coloring
- **Reload pass to fix spills (allocator does not generate spill code)**

gcc's approach, unless -fnew-ra

- **Can also do global then local (Morgan)**
- **Advantages? Disadvantages?**

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 30

Summary

```

graph TD
    Build --> Simplify
    Simplify --> Coalesce
    Coalesce --> PotentialSpill[Potential Spill]
    PotentialSpill --> Simplify
    PotentialSpill --> Coalesce
    PotentialSpill --> Select
    Select --> ActualSpill[Actual Spill]
    ActualSpill --> Build
    
```

- Spilling
- Spill Selection
- Special Registers
- Live Range Splitting
- Alternative Algorithms

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 31

What's Next

Project Proposals

Instruction Scheduling

- Compiling for multi-issue processors

CS745: Register Allocation © David Ryan Koes & Seth Copen Goldstein & Todd C. Mowry 2002-3 32