Register Allocation
Outline

- What is register allocation
- Webs
- Interference Graphs
- Graph coloring
- Spilling
- Splitting
- More optimizations
Storing values between def and use

• 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
Register Allocation

• Deciding which values to store in 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
    • Limited mem-to-mem ALU ops, may need two instructions
  – Probably is 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
Issues

• Fewer instructions when using registers
  – Additional instructions when using 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 if program characteristics were different

• But only a small number of registers available
  – Usually 16 integer and 16 floating-point registers
  – Some of those registers have fixed users (ex: RSP, RBP)
Outline

• What is register allocation
• Key ideas in register allocation
• Webs
• Interference Graphs
• Graph coloring
• Splitting
• More optimizations
Summary of Register Allocation

• You want to put each temporary in a register
  – *But*, you don’t have enough registers.

• Key Ideas:
  1. When a temporary goes dead, its register can be reused
  2. Two live temporaries can’t use the same register at the same time
Summary of Register Allocation

- When a temporary goes dead, its register can be reused
- Example:
  
  ```
  a := c + d  
  e := a + b  
  f := e - 1  
  r1 := c + d  
  r1 := r1 + b  
  r1 := r1 - 1  
  ```

  (assume that a and e die after use)

- temporaries a, e and f can go in the same register
Summary of Register Allocation

• Two live temporaries can’t use the same register at the same time

• Example 2:
  
  a := c + d  
e := a + b  
f := e - a  

• temporaries e and a can not go in the same register
  
r1 := c + d  
r2 := r1 + b  
r1 := r2 – r1
When things don’t work out

• Sometimes more live variables than registers
  
  a := c + d  
  e := c + b  
  f := e – c  
  g := e + f  
  h := a + g  

  Won’t work for 2 registers

  (assume only g and h live at the end)

• You can split a live range by storing to memory
  
  a := c + d  
  store a  
  e := c + b  
  f := e – c  
  g := e + f  
  load a  
  h := a + g
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 gen)
Outline

• What is register allocation
• Key ideas in register allocation
• Webs
• Interference Graphs
• Graph coloring
• Splitting
• More optimizations
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
Example

```
def x
  def y
use x
use y
```

```
def x
use x
def x
use y
```

```
def x
```

```
de```
Example

def x
def y

use x
use y

use x
use y

def x

use x

Saman Amarasinghe & Armando Solar-Lezama
©MIT 6.035
Example
Example
Example
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
Outline

• What is register allocation
• Webs
• Interference Graphs
• Graph coloring
• Splitting
• More 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 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
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
Example

Webs s1 and s2 interfere
Webs s2 and s3 interfere

Saman Amarasinghe & Armando Solar-Lezama

©MIT 6.035
Outline

- Overview of procedure optimizations
- What is register allocation
- A simple register allocator
- Webs
- Interference Graphs
- Graph coloring
- 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
Graph Coloring Example

- 1 Color
Graph Coloring Example
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
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)
  - 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)
Coloring Example

\[ N = 3 \]
Coloring Example

\[ N = 3 \]

\begin{itemize}
  \item s1
  \item s2
  \item s0
  \item s3
  \item s4
\end{itemize}
Coloring Example

$N = 3$

$s1 \quad s2 \quad s0 \quad s3 \quad s4$
Coloring Example

\[ N = 3 \]

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

\[ N = 3 \]

\[ \begin{array}{cc}
   & s1 \\
 s0 & s2 \\
   & s3
\end{array} \]

\[ s1 \quad s2 \quad s4 \]

Saman Amarasinghe & Armando Solar-Lezama 50

© MIT 6.035
Coloring Example

\( N = 3 \)

\[ s_1 \quad s_2 \quad s_3 \quad s_4 \]

\[ s_0 \]

\( s_1, s_2, s_3, s_4 \)
Coloring Example

\[ N = 3 \]

\begin{align*}
\text{s1} & \quad \text{s2} \\
\text{s3} & \quad \text{s4} \\
\text{s0} &
\end{align*}
Coloring Example

$N = 3$

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

N = 3

\begin{align*}
s_0 & \quad \text{Blue} \\
s_1 & \quad \text{Green} \\
s_2 & \quad \text{Brown} \\
s_3 & \\
s_4 &
\end{align*}
Coloring Example

\[ N = 3 \]

\[ s_0 \]

\[ s_1 \]

\[ s_2 \]

\[ s_3 \]

\[ s_4 \]

\[ s_1 \]

\[ s_2 \]

\[ s_4 \]
Coloring Example

\[ N = 3 \]

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

\[ N = 3 \]

\[ s0 \quad s1 \quad s2 \quad s3 \quad s4 \]
Coloring Example

\( N = 3 \)

\begin{align*}
&\text{s1} & \text{s2} \\
&\text{s3} & \text{s4} \\
\end{align*}
Coloring Example

\[ N = 3 \]

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

$N = 3$
Coloring Example

\[ N = 3 \]

\[ \begin{align*}
   s_1 & \quad \text{blue} \\
   s_2 & \quad \text{green} \\
   s_0 & \quad \text{brown} \\
   s_3 & \quad \text{green} \\
   s_4 & \quad \text{brown}
\end{align*} \]
Another Coloring Example

\[ N = 3 \]
Another Coloring Example

\[ N = 3 \]

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

\[ N = 3 \]
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{s1} \\
\text{s0} \\
\text{s2} \\
\text{s3} \\
\text{s4}
\end{array}

\begin{array}{c}
s2 \\
s3 \\
s4
\end{array}
Another Coloring Example

\[ N = 3 \]

- \( s_0 \)
- \( s_1 \)
- \( s_2 \)
- \( s_3 \)
- \( s_4 \)
Another Coloring Example

\[ N = 3 \]

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

\[ N = 3 \]

\[ \begin{align*}
  s_0 & \quad \text{Blue} \\
  s_1 & \quad \text{Green} \\
  s_2 & \quad \text{Brown} \\
  s_3 & \quad \text{Brown} \\
  s_4 & \quad \text{Yellow} 
\end{align*} \]
Another Coloring Example

N = 3 ⬇️

s1
s2
s3
s0
s4

©MIT
Another Coloring Example

$N = 3$

- $s_1$
- $s_2$
- $s_0$
- $s_3$
- $s_4$
Another Coloring Example

\[ N = 3 \]

s0

s1

s2

s3

s4

©MIT
What Now?

• 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
Which web to pick?

- 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
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 Compute Spill Cost

- Goal: give priority to values used in loops
- So assume loops execute 10 or 100 times
- Spill cost =
  - sum over all def sites of cost of a store instruction times 10 to the loop nesting depth power, plus
  - sum over all use sites of cost of a load instruction times 10 to the loop nesting depth power
- Choose the web with the lowest spill cost
With 1 Register, Which Variable Gets Spilled?

Spill Cost Example

- Spill Cost For x
  - storeCost + loadCost

- Spill Cost For y
  - 9*storeCost + 9*loadCost

def x
def y

use y
def y

use x
use y
Outline

• Overview of procedure optimizations
• What is register allocation
• A simple register allocator
• Webs
• Interference Graphs
• Graph coloring
• Splitting
• More 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
Splitting Example

```
def z
use z

def x
def y
use x
use x
use y

use z
```
Splitting Example

def z
use z

def x
def y
use x
use y

use z

x y z

x

y

z
Splitting Example

```
def z
use z
```

```
def x
def y
use x
use y
```

```
use z
```

```
x y z
```

```
x
use x
use x
use y
```

```
y
```

```
z
```

2 colorable?
Splitting Example

2 colorable? NO!
Splitting Example

def z
use z

def x
def y
use x
use y
use x
use y

use z
Splitting Example

def z
use z

def x
def y
use x
use x
use y

use z
Splitting Example

def z
  use z

def x
def y
  use x
  use y

use z

x y z

z1

x

y

z2
Splitting Example

```
def z
  use z
```

```
def x
  def y
  use x
  use y
use z
```

```
x y z
```

```
z1
x y z2
```

2 colorable?
Splitting Example

def z
use z

def x
def y
use x
use y

use z

2 colorable? YES!
Splitting Example

def z
use z

def x
def y
use x
use y
use z

x y z
r1 r1 r2

x
y
z1
z2

2 colorable?
YES!
Splitting Example

def z
use z
str z

def x
def y
use x
use x
use y

ld z
use z

x y z

r1 r1 r1

r2

z1 z2

x y

2 colorable? YES!
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
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 ration to spill
Outline

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

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

• Find register copy instructions sj = si

• If sj and si 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 a given time
  - first 6 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 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)
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
Summary

• Register Allocation
  – Store values in registers between def and use
  – Can improve performance substantially

• Key concepts
  – Webs
  – Interference graphs
    Colorability
  – Splitting