

Compiler Construction 2012/2013

Register Allocation for Programs in SSA-Form

Peter Thiemann

February 11, 2013

Outline

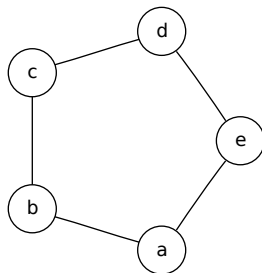
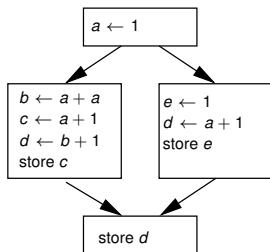
- 1 Motivation
- 2 Foundations
- 3 Spilling
- 4 Coloring
- 5 Coalescing
- 6 Register Constraints
- 7 Conclusion

Foundation: Sebastian Hack, Daniel Grund, Gerhard Goos.
Towards Register Allocation for Programs in SSA-Form. 2005.

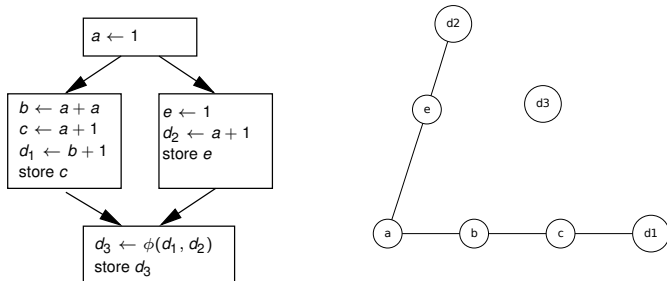
- register allocation maps temporaries to physical registers such that their live ranges do not interfere
- common technique: graph coloring [Chaitin] of the interference graph

Example: Program and its Interference Graph

Three Registers Needed



Example Program in SSA Form



- Two registers available: but copy instruction needed
- Three registers available: use all and eliminate copy

SSA and Register Allocation

- ϕ -functions replaced by moves before register allocation
- moves lead to coalescing
- may lead to spill

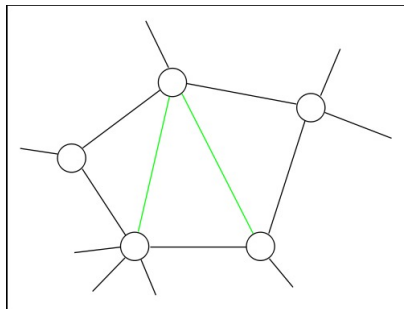
Background

- any undirected graph occurs as inference graph of a program
- finding a minimal k -coloring of a general graph is NP-complete
- hence, the heuristic feedback algorithm Build \rightarrow Coalesce \rightarrow Color \rightarrow Spill? required
- [coalescing changes colorability of graph]

Background Graph Theory

Definition

In a chordal graph, every cycle of four or more nodes has a chord, i.e., an edge between two of the nodes that does not belong to the cycle. (Also: triangulated graph)



source: <http://upload.wikimedia.org/wikipedia/commons/thumb/3/34/Chordal-graph.svg/>

220px-Chordal-graph.svg.png

Definition

- Clique: fully connected subgraph.
- Clique number $\omega(G)$: Size of largest clique of G .
- Chromatic number $\chi(G)$: Minimum k such that G is k -colorable.
- In a perfect graph, the chromatic number of each induced subgraph is equal to the size of its largest clique.

- In a perfect graph, graph coloring can be solved in polynomial time.

Graph Coloring and SSA Form

- Interference graphs of SSA programs are chordal graphs
see also [Pereira&Palsberg 2005] [Brisk 2005]
[Bouchez,Darte&Rastello 2005]
- ⇒ spilling and coalescing can be decoupled
- Every chordal graph is a perfect graph
- ⇒ number of registers needed = size of largest clique
the largest set of variables that are live at the same time
- ⇒ Spilling can be performed once and for all before register allocation

Graph Coloring and SSA Form

Continued

- Coloring a chordal graph takes $O(|V|^2)$
- Given the dominator tree and the live ranges, then coloring takes $O(\omega(G) \cdot n)$ time
 - n number of instructions
 - $\omega(G)$ size of largest clique in G
 \leq number of registers after spilling
- Usually, ϕ -functions \mapsto move instructions
- Early coalescing is harmful
- Instead of coalescing, try to assign the same color

Outline

- 1 Motivation
- 2 Foundations**
- 3 Spilling
- 4 Coloring
- 5 Coalescing
- 6 Register Constraints
- 7 Conclusion

ϕ -functions

- ϕ -functions are not functions, but a notational device
- ϕ -functions do not cause interference
- There is no ordering among different ϕ -functions at the beginning of a block; ideally, they should “evaluate” simultaneously

⇒ different notation

$$\begin{array}{l} y_1 \leftarrow \phi(x_{11}, \dots, x_{1n}) \\ \vdots \\ y_m \leftarrow \phi(x_{m1}, \dots, x_{mn}) \end{array} \quad \Longrightarrow \quad \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \leftarrow \Phi \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \dots & x_{mn} \end{bmatrix}$$

Interference Graphs of SSA Programs

Let \mathcal{D}_v be the node defining v .

Lemma 1

If two registers v and w are live at node n , then either \mathcal{D}_v dominates \mathcal{D}_w or \mathcal{D}_w dominates \mathcal{D}_v .

Lemma 2

If v and w interfere and \mathcal{D}_v dominates \mathcal{D}_w , then v is live at \mathcal{D}_w .

Lemma 3

Let (u, v) and (v, w) be edges in the interference graph, but not (u, w) .

If \mathcal{D}_u dominates \mathcal{D}_v , then \mathcal{D}_v dominates \mathcal{D}_w .

Interference Graphs of SSA Programs are Chordal

Proof

Suppose there is a chain of length $n \geq 4$ in the interference graph:

$$x_1 - x_2 - \dots - x_i - \dots - x_n$$

where there is no edge between x_i and x_j , for $1 \leq i < j < n$ and $j - i > 1$.

Assume that $\mathcal{D}_{x_1} \text{ dom } \mathcal{D}_{x_2}$. By induction, using Lemma 3, $\mathcal{D}_{x_i} \text{ dom } \mathcal{D}_{x_{i+1}}$, for $1 \leq i < n$. Suppose further that there is an edge $x_1 - x_n$. Hence, there is some block ℓ where x_1 and x_n are live and ℓ must be dominated by all \mathcal{D}_{x_i} , for $1 \leq i \leq n$. For each x_i ($i > 1$) there is a path from \mathcal{D}_{x_i} to ℓ , which does not go through \mathcal{D}_{x_1} . Hence, the edge $x_1 - x_i$ must be in the graph. Contradiction.

Outline

- 1 Motivation
- 2 Foundations
- 3 Spilling**
- 4 Coloring
- 5 Coalescing
- 6 Register Constraints
- 7 Conclusion

- Problem: the interference graph does not reflect the number of uses of a register
- ⇒ \exists work to break the live ranges in smaller pieces
- [Bouchez 2005] shows that “splitting live ranges to lower the register pressure to a fixed k while inserting a minimum number of reload instructions is NP-complete”

A Foundation for Spilling

Lemma

For each clique $C \subset G$ with $V_C = \{v_1, \dots, v_n\}$, there is a permutation $\sigma : V_C \rightarrow V_C$ such that $\mathcal{D}_{\sigma(v_i)}$ dominates $\mathcal{D}_{\sigma(v_{i+1})}$ for $1 \leq i < n$.

Theorem

Let G be the interference graph of an SSA program and C be an induced subgraph of G . C is a clique in G iff there exists a label in the program where all V_C are live.

Spilling with Belady's Algorithm

- Let ℓ be a node where $l > k$ variables are live
- Belady's algorithm spills those $l - k$ variables whose uses are farthest away (in minimum number of instructions executed) from ℓ .

$$\text{nextuse}(\ell, v) = \begin{cases} \infty & \text{if } v \text{ not live at } \ell \\ 0 & \text{if } v \text{ used at } \ell \\ 1 + \min_{\ell' \in \text{succ}[\ell]} \text{nextuse}(\ell', v) & \text{otherwise} \end{cases}$$

- Apply Belady's algorithm to each basic block

Belady's Algorithm for Basic Blocks

- Let P be the set of variables passed into block B : the variables live-in at B and the results of the ϕ -functions
 - Let $\sigma : P \rightarrow P$ be a permutation which sorts P ascendingly according to *nextuse*
- ⇒ Pass the set of variables $I = \{p_{\sigma(1)}, \dots, p_{\sigma(\min(k, |I|))}\}$ in registers
- Traverse the nodes in a basic block from entry to exit.
 - Let Q be the set of all variables currently in registers ($|Q| \leq k$, initially $Q \leftarrow I$)

Belady's Algorithm for Basic Blocks

continued

- At an instruction

$$\ell : \underbrace{(y_1, \dots, y_m)}_{\mathcal{D}_\ell} \leftarrow \tau \left(\underbrace{x_1, \dots, x_n}_{\mathcal{U}_\ell} \right)$$

set $R \leftarrow \mathcal{U}_\ell \setminus Q$

- if $R \neq \emptyset$, then
 - reloads have to be inserted and $\max(|R| + |Q| - k, 0)$ variables are removed from Q
 - remove those with highest *nextuse*
- If $v \in I$ is displaced before used, then v need not be passed to B in a register
- Let in_B be the set $v \in I$ which are used in B before they are displaced.

Belady's Algorithm for Basic Blocks

continued

- τ displaces $\max(|\mathcal{D}_\ell| + |Q| - k, 0)$ variables from Q
- To decide which variables to displace we use

$$\text{nextuse}'(\ell, v) = 1 + \min_{\ell' \in \text{succ}[\ell]} \text{nextuse}(\ell', v)$$

- Let out_B be the set Q after processing the last node in a block

Belady's Algorithm Extended

- To connect the blocks, ensure that each variable in in_B is in a register on entry to B .
- At the end of each predecessor P' of B insert reloads for all $in_B \setminus out_{P'}$ (recall edge splitting)

Outline

- 1 Motivation
- 2 Foundations
- 3 Spilling
- 4 Coloring**
- 5 Coalescing
- 6 Register Constraints
- 7 Conclusion

Coloring Chordal Graphs

- perfect elimination orders (PEO)
- order in which variables are removed from graph
- basis: simplicial nodes (all neighbors belong to the same clique)
- **Lemma:** Each chordal graph has a simplicial node
- Removing a node from a chordal graph preserves chordality
- PEOs are related to the dominance tree

Coloring Chordal Graphs

Theorem

An SSA variable v can be added to a PEO of G if all variables whose definitions are dominated by the definition of v have been added to the PEO.

Proof

For a contradiction, assume v is not simplicial. Hence, v has two neighbors a and b which are not connected.

As all variables whose definitions are dominated by \mathcal{D}_v are already part of the PEO and removed, it must be that \mathcal{D}_a dominates \mathcal{D}_v . By a previous lemma, \mathcal{D}_v dominates \mathcal{D}_b , contradicting the assumption.

Coloring Chordal Graphs

COLORPROGRAM (Program P)

COLORRECURSIVE (start block of P)

COLORRECURSIVE (Basic block B)

$assigned \leftarrow$ colors of the live-in(B)

for each instruction $(b_1, \dots, b_m) \leftarrow \tau(a_1, \dots, a_n)$ from entry to exit **do**

for $a \in \{a_1, \dots, a_n\}$ **do**

if last use of a **then**

$assigned \leftarrow assigned \setminus color(a)$

for $b \in \{b_1, \dots, b_n\}$ **do**

$color(b) \leftarrow$ one of $allcolors \setminus assigned$

for each C where $B = idom(C)$ **do**

COLORRECURSIVE(C)

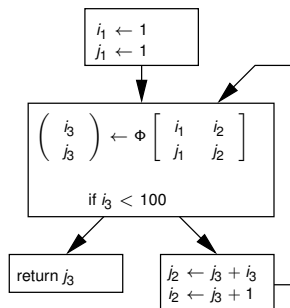
Outline

- 1 Motivation
- 2 Foundations
- 3 Spilling
- 4 Coloring
- 5 Coalescing**
- 6 Register Constraints
- 7 Conclusion

Coalescing Phase

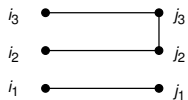
- Goal: minimize number of copy/move instructions
- Causes of copy/move instructions
 - ϕ -functions
 - register constraints of target architecture (pre-colored nodes)

Implementation of ϕ -functions



- Seems to require two registers
- However, implementing Φ by the moves $i_3 \leftarrow i_2; j_3 \leftarrow j_2$ creates an interference between i_3 and j_2

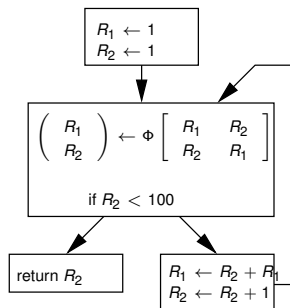
Interference from Implementation of Φ



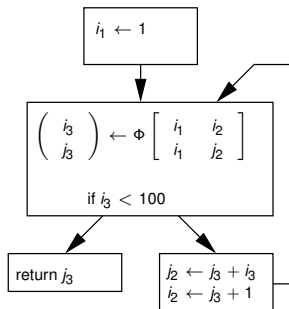
Removal of Φ without Using Extra Registers

- Consider $(b_1, \dots, b_n) \leftarrow \sigma(a_1, \dots, a_n)$
- A multi-assignment that permutes the contents of the registers according to σ
- For the example program, a permutation is needed that swaps two registers:

Example Program After Register Assignment

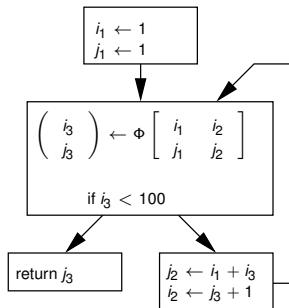


Example Where Copying is Needed



- Φ duplicates i_1 into i_3 and j_3

Example Where Copying is Needed



- i_1 interferes with Φ

Duplication in the Removal of Φ

- Duplication (i.e., extra registers) are only needed if
 - a Φ argument is used multiple times in one column
 - a Φ argument is live-in at the block of Φ
- Interference with a value defined by Φ does not require duplication.

Implementation of Permutations

Register swaps Swap instructions of the processor;

xor trick: $a \leftarrow a \oplus b; b \leftarrow a \oplus b; a \leftarrow a \oplus b$

Moves assuming a free backup register, each cycle C can be implemented with $|C| + 1$ move instructions for example, `$at` in MIPS

Optimizing Φ -functions

- The cost of implementation for a permutation σ is related to the number of fixpoints of σ
- Variable x is a fixpoint if

$$(\dots, x', \dots) = \sigma(\dots, x, \dots)$$

and x and x' are assigned the same register

\Rightarrow no code needs to be generated for a fixpoint

Optimizing Φ -functions

Problem Statement

$$\ell : \begin{pmatrix} y_1 \\ \vdots \\ y_m \end{pmatrix} \leftarrow \Phi \begin{bmatrix} x_{11} & \dots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \dots & x_{mn} \end{bmatrix}$$

Given a k -coloring $f : V \rightarrow \{1, \dots, k\}$ define the cost of p by

$$c_f(\ell) = \sum_{i=1}^m \sum_{j=1}^n \text{cost}_f(y_i, x_{ij})$$

where $\text{cost}_f(a, b) = \begin{cases} w_{ab} & \text{if } f(a) \neq f(b) \\ 0 & \text{otherwise} \end{cases}$ with $w_{ab} \geq 0$ the cost

of copying b to a .

The overall cost of a program P under coloring f is

$$c(P, f) = \sum_{\ell \text{ is } \Phi\text{-node}} c_f(\ell)$$

Optimizing Φ -functions

Problem Statement

SSA-Maximize-Fixed-Points

Given an SSA program P and its interference graph G . Find a coloring f of G for which $c(P, f)$ is minimal.

Theorem

SSA-Maximize-Fixed-Points is NP-complete.

Heuristics for Optimizing Φ -functions

- Start with a k -coloring
- Modify color assignments to lower the cost
Non-local changes in the coloring may be required!
- A valid k -coloring is always maintained
- For each row i of the Φ -function

$$\begin{pmatrix} p_1 \\ \vdots \\ p_m \end{pmatrix} \leftarrow \Phi \begin{bmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix}$$

define an optimization unit (OU) consisting of p_i and all a_{ij} that do not interfere with p_j (at least one)

Perm-Optimizer

COALESCE(G)

$pinned \leftarrow \emptyset$

for each OU (p, a_1, \dots, a_k) **do**

for each color c assignable to p **do** {Init}

$C_c \leftarrow G[p, a_1, \dots, a_k]$ { conflict graph }

$S_c \leftarrow$ max weighted stable subset of C_c {weight of a_i is w_{pa_i} }

 Insert (c, C_c, S_c) in min-queue Q { ordered by $w(S_c)$ }

repeat { Test }

$candidates \leftarrow \emptyset$

$g \leftarrow f$ {copy the current coloring}

 pop (c, C, S) from Q

$C' \leftarrow$ TEST(c, C, S)

if $C' \neq \text{nil}$ **then**

$S' \leftarrow$ maximum weighted stable subset of C'

 Insert (c, C', S') into Q

until $C' = \text{nil}$

if $|candidates| > 1$ **then**

$pinned \leftarrow pinned \cup candidates$

$f \leftarrow g$ { update coloring }

```
TEST( $c, C, S$ )  
  {  $S = \{p, a_1, \dots, a_l\}$  processed in this order }  
  for  $u \in S$  do  
    ( $s, v$ )  $\leftarrow$  TRYCOLOR( $u, \text{nil}, c$ )  
    if  $s = \text{ok}$  then  
       $\text{candidates} = \text{candidates} \cup \{u\}$   
    else if  $s = \text{candidate}$  and  $v \neq p$  then  
      return ( $V_C, E_C \cup \{(v, u)\}$ )  
    else  
      return ( $V_C, E_C \cup \{(u, u)\}$ )  
  return nil
```

Perm-Optimizer III

```
TRYCOLOR( $v \in V_G, u \in V_G, c$ )  
   $c_v \leftarrow g(v)$   
  if  $c = c_v$  then  
    return (ok, nil)  
  else if  $v \in \text{pinned}$  then  
    return (pinned,  $v$ )  
  else if  $v \in \text{candidates}$  then  
    return (candidate,  $v$ )  
  else if  $c$  is not allowed for  $v$  then  
    return (forbidden,  $v$ )  
  for each  $n$  with  $(v, n) \in E_G, n \neq u, g(n) = c$  do  
    { try to swap colors with neighbor }  
     $(s, v') \leftarrow \text{TRYCOLOR}(n, v, c_v)$   
    if  $s \neq \text{ok}$  then  
      return ( $s, v'$ )  
   $g(v) \leftarrow c$   
  return (ok, nil)
```

Outline

- 1 Motivation
- 2 Foundations
- 3 Spilling
- 4 Coloring
- 5 Coalescing
- 6 Register Constraints**
- 7 Conclusion

Register Constraints

- Most processor architectures have instructions where the operands are restricted to specific registers
 - Graph coloring approach
 - 1 split live range at constraining definition
 - 2 add one pre-colored node for each register
 - 3 connect definition with all pre-colored nodes, except the one with the required color
 - For chordal graphs, coloring is in P iff each color is used only once in pre-coloring.
Unrealistic constraint for register allocation
- ⇒ Delegate to the Perm-Optimizer

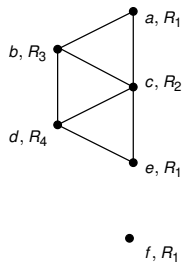
Register Constraints by Perm-Optimization

- Insert $(a'_i) = \Phi[a_i]$ (for all live registers) in front of each instruction with register constraints
- ⇒ all live variables can change register at that point
- ⇒ interference graph breaks in two unconnected components
- ⇒ each color occurs only once as pre-coloring in each component
- first do coloring, then Perm-Optimization

Example Register Constraints

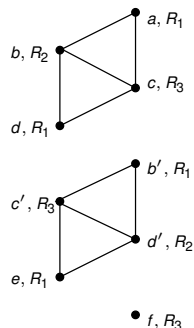
Code and Colored Interference Graph

$a_{R_1} \leftarrow \dots$
 $b \leftarrow \dots$
 $c \leftarrow b + 1$
 $d \leftarrow a + 1$
 $e_{R_1} \leftarrow b + c$
 $f \leftarrow c + d$
 \vdots



Example Register Constraints with Φ Inserted

Code and Colored Interference Graph

$$\begin{aligned} a_{R_1} &\leftarrow \dots \\ b &\leftarrow \dots \\ c &\leftarrow b + 1 \\ d &\leftarrow a + 1 \\ \begin{pmatrix} b' \\ c' \\ d' \end{pmatrix} &\leftarrow \Phi \begin{bmatrix} b \\ c \\ d \end{bmatrix} \\ e_{R_1} &\leftarrow b' + c' \\ f &\leftarrow c' + d' \\ &\vdots \end{aligned}$$


Outline

- 1 Motivation
- 2 Foundations
- 3 Spilling
- 4 Coloring
- 5 Coalescing
- 6 Register Constraints
- 7 Conclusion**

Conclusion

- Interference graphs for SSA programs are chordal
- ⇒ main phases of register allocation (spilling, coloring, coalescing) can be decoupled
- Procedure for spilling based on the correspondence live sets \leftrightarrow cliques in interference graph (without constructing the graph)
- (Optimal spilling via ILP solving)
- Optimal coloring in linear time (w/o constructing the graph)
- Optimal coalescing is NP-complete
 - Heuristic
 - (Optimal coalescing via ILP solving)
- Register constraints expressible

Alternatives

- [Pereira&Palsberg, APLAS 2005] observe that 95% of the methods in the Java 1.5 library give rise to chordal interference graphs and give an algorithm for register allocation under this assumption
- [Pereira&Palsberg, PLDI 2008] give a general, industrial strength framework for register allocation based on puzzle solving. It first transforms its input to elementary programs, a strengthening of SSA programs.
- [Pereira&Palsberg, CC 2009] propose a different, spill-free way to perform SSA elimination after register coloring
- [Pereira&Palsberg, CC 2010] present Punctual Coalescing, a scalable, linear time, locally optimal algorithm for coalescing.
- [Hack&Good, PLDI 2008] register coalescing by graph recoloring.
- [Braun&Hack, CC 2009] present an improved spilling algorithm for programs in SSA form.