

SSA Register Allocation

Sebastian Hack, Daniel Grund, Gerhard Goos

(hack|ggoos)@ipd.info.uni-karlsruhe.de grund@cs.uni-sb.de

Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe

Sommersemester 2006



Reading

Sebastian Hack, Daniel Grund, Gerhard Goos

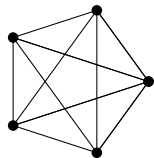
Towards Register Allocation for Programs in SSA-form

Universität Karlsruhe, Technical Report 2005-27, September 2005

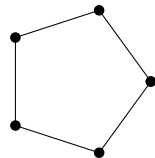
http://www.info.uni-karlsruhe.de/~hack/ra_ssa.pdf



Complete Graphs and Cycles



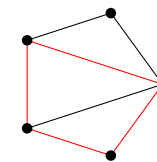
Complete Graph K^5



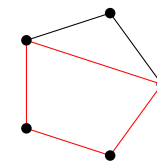
Cycle C^5



Induced Subgraphs



Graph with a C^4 subgraph



Graph with a C^4 induced subgraph

Note

Induced complete graphs are called cliques



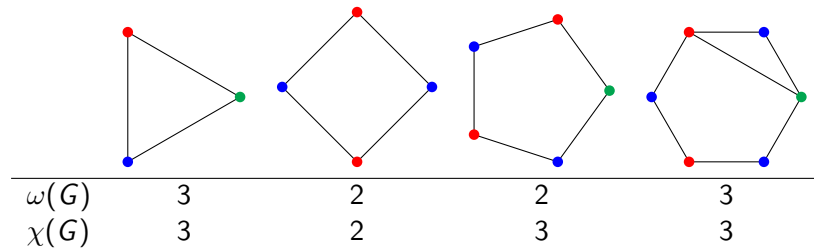
Clique number and Chromatic number

$\omega(G)$ Size of the largest clique in G

$\chi(G)$ Number of colors in a minimum coloring of G

Corollary

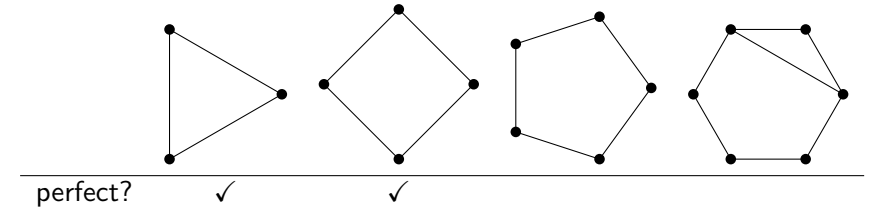
$\omega(G) \leq \chi(G)$ holds for each graph G



Perfect Graphs

Definition

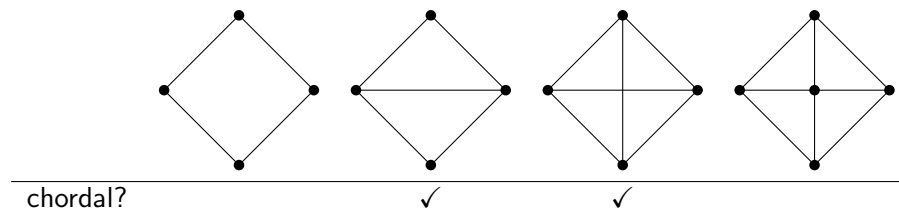
G is perfect $\iff \chi(H) = \omega(H)$ for each induced subgraph H of G



Chordal Graphs

Definition

G is chordal $\iff G$ contains no induced cycles longer than 3



Theorem

Chordal graphs are perfect

Theorem

Chordal graphs can be colored optimally in $O(|V| \cdot \omega(G))$



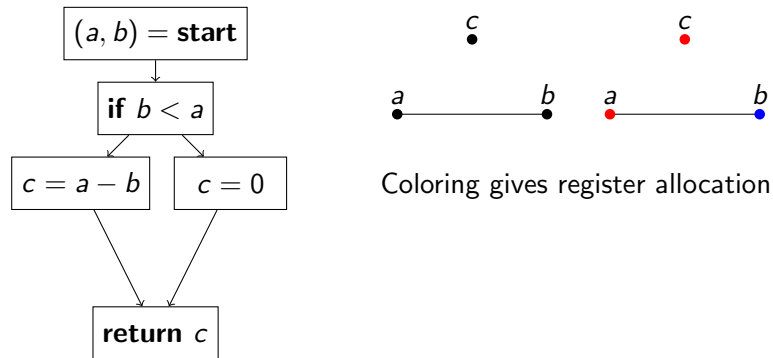
Register Allocation

- Register Allocation is the task of mapping the program's variables to processor registers
- Issues to be covered:
 - **Spilling** Put variables into memory if there are not enough registers
 - **Coalescing** Eliminate unnecessary copies in the program
- Often reduced to graph coloring

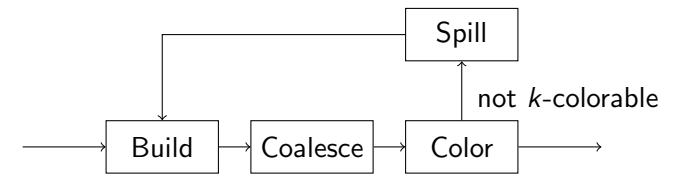


Interference Graphs

- Two variables are live at the same label ▶ they **interfere**
- Each variable has a node in the interference graph (IG)
- Whenever two variables interfere, there is an edge between the corresponding nodes



Chaitin/Briggs Register Allocator



- Every undirected graph can occur as an interference graph
- Determining chromatic number is NP-complete
- Color using heuristic \Rightarrow Iteration necessary
- Spilling is focused on the graph



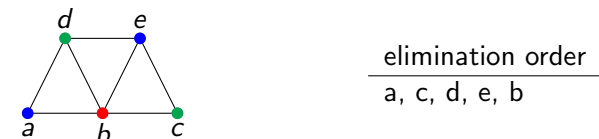
Coloring

- Subsequently remove the nodes from the graph
- Re-insert the nodes in reverse order
- Assign each node the next possible color



Perfect Elimination Orders

- Suppose all (not yet eliminated) neighbors of a node n form a clique



Theorem (from graph theory)

- A PEO allows for an optimal coloring in **polynomial** time
- The number of colors is bound by the size of the largest clique



Theorem (from graph theory)

For each graph there is an elimination order leading to an optimal coloring

Perfect Elimination Orders

- Graphs with holes larger than 3 have no PEOs, e.g.



- Graphs with PEOs are called **chordal**
- Chordal graphs are **perfect**, i.e.

$$\chi(H) = \omega(H) \text{ for each } H \subseteq G$$

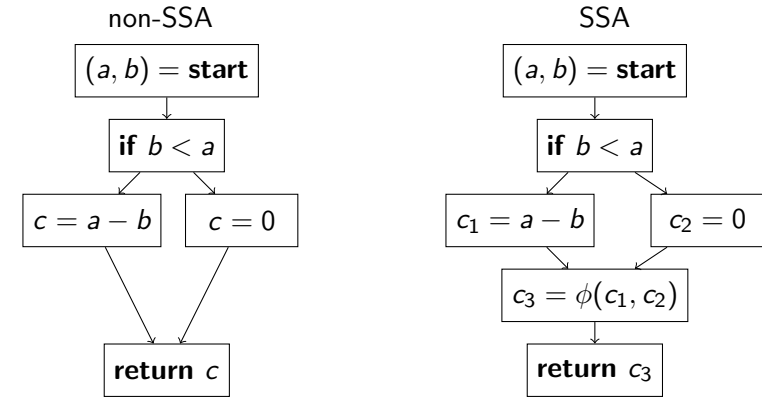
Theorem

- The dominance relation in SSA programs induces a PEO in the IG
- Thus, SSA IGs are chordal



SSA-Form

- Each variable has exactly one definition
 - Identity of variables and dynamic constants (values)
- ϕ -operations select values dependent on control flow

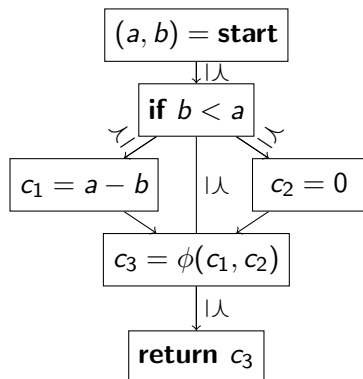


Dominance

Crucial for SSA-form programs is the concept of dominance:

Definition

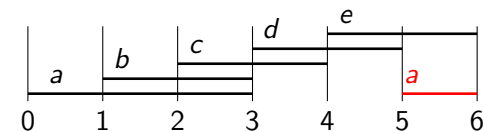
l_1 dominates l_2 if each path from **start** to l_2 goes through l_1



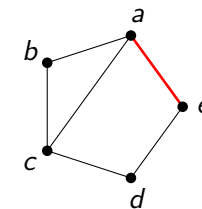
- Each node has a unique **immediate dominator**
- Thus, dominance induces a tree on the control flow graph
- Thus, dominance is also a partial order



Why are SSA IGs chordal? — Intuition



- Each interval corresponds to a lifetime of a variable
 - a node in the interference graph



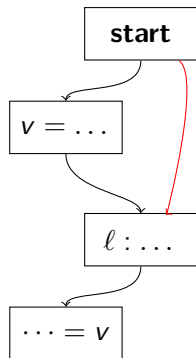
- Can we make a cycle by drawing an edge from a to e ?
- Only by letting a “start again” at 5



Liveness and Dominance

Budimčić, PLDI '02

- Each label where a value v is live, is dominated by v 's definition.
- We write shortly: $\mathcal{D}_v \preceq \ell$



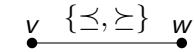
Proof by contradiction

- Assume ℓ is not dominated by \mathcal{D}_v
- Then there's a path from **start** to some usage of v **not** containing the definition of v
- This cannot be since each value must have been defined before it is used



Interference and Dominance I

- Assume v, w **interfere**, i.e. they are live at some label ℓ
- Then, $\mathcal{D}_v \preceq \ell$ and $\mathcal{D}_w \preceq \ell$
- Since dominance is a tree, either $\mathcal{D}_v \preceq \mathcal{D}_w$ or $\mathcal{D}_w \preceq \mathcal{D}_v$



Consequences

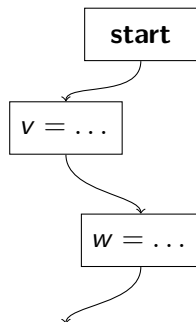
- Each edge in the interference graph has a direction according to dominance
- The interference graph is an "excerpt" of the dominance relation



Interference and Dominance II

Budimčić, PLDI '02

- Assume $v \xrightarrow{\preceq} w$
- Then, v is live at \mathcal{D}_w



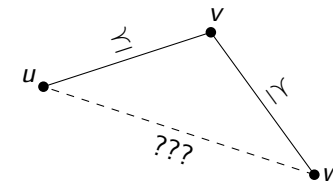
Proof by contradiction

- Assume v was not live at \mathcal{D}_w
- Then there is no path from \mathcal{D}_w to any use of v
- So v and w do not interfere \downarrow



Interference and Dominance III

- Consider three nodes u, v, w in the IG:



- u is live at \mathcal{D}_v
- w is live at \mathcal{D}_v
- Thus, they interfere

Conclusion

All values

- interfering with v
 - whose definitions dominate the one of v
- are members of **the same clique**



Dominance and PEOs

- Before a value v is added to a PEO, add all values whose definitions are dominated by \mathcal{D}_v
- Thus, a post order walk of the dominance tree defines a PEO
- IGs of SSA-form programs can be colored in $O(\chi(G) \cdot |V|)$
- Ideally without constructing the interference graph itself



Spilling

Consequences

- The chromatic number of the IG is **exactly** determined by the number of live variables at the labels
- Lowering the number of values live at each label to k makes the IG k -colorable
- We know in advance where values must be spilled
 - ▶ All labels where the pressure is larger than k
- Spilling can be done before coloring **and**
- coloring will always succeed afterwards



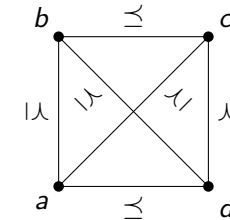
Conclusion

- No iteration as in Chaitin/Briggs-allocators
- Interference graph has to be built only once (if at all)

Spilling

Theorem

For each clique in the IG there is a label in the program where all nodes in the clique are live.



- Dominance induces a **total order** inside the clique
⇒ There is a “greatest” value d
- All others are live at the definition of d



Getting out of SSA

- We now have a k -coloring of the SSA interference graph
- Can we turn it into a valid register allocation using k registers for the corresponding non-SSA program?

Central question

How to handle ϕ -functions?



φ-Functions

- All φ-functions in a basic block

$$\begin{aligned}
 y_1 &\leftarrow \phi(x_{11}, \dots, x_{n1}) \\
 &\vdots \\
 y_m &\leftarrow \phi(x_{1m}, \dots, x_{nm})
 \end{aligned}$$

execute **simultaneously before** all other instructions in that block

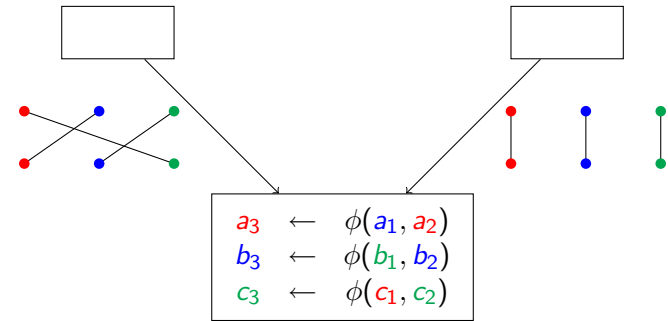
- Arriving from the *i*-th edge, the φ-functions work as a **parallel copy**

$$(y_1, \dots, y_m) \leftarrow (x_{i1}, \dots, x_{im})$$



φ-Functions

- Consider following example

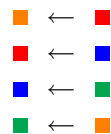


- The φs represent register permutations on the control flow edges

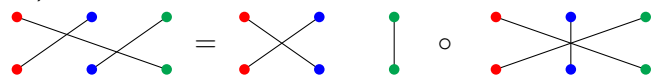


Permutations

- A permutation can be implemented with copies if one auxiliary register is available



- Permutations can be implemented by a series of transpositions (i.e. swaps)



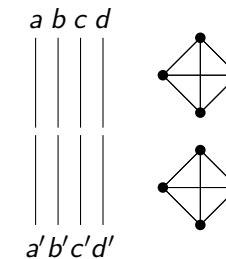
- A transposition can be implemented by three xors **without** a third register



Difference to classical SSA-destruction

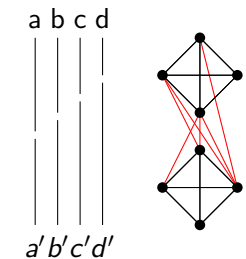
Parallel copies

$$(a', b', c', d') \leftarrow (a, b, c, d)$$



Sequential copies

$$\begin{aligned}
 d' &\leftarrow d \\
 c' &\leftarrow c \\
 b' &\leftarrow b \\
 a' &\leftarrow a
 \end{aligned}$$



Coalescing

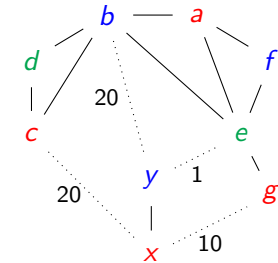
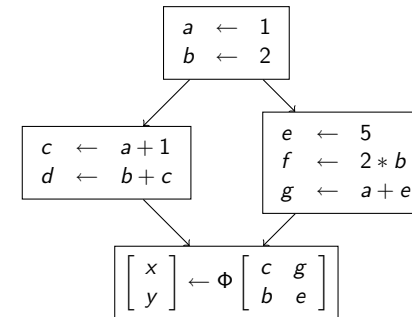
- Minimize number of instructions to be inserted for all ϕ s
- Current practice: Merging nodes in the IG to avoid copies
 - Renders the graph unchordal
 - Lose information on the chromatic number
 - When done aggressively, introduce spills in favor of (eliminated) copies



Coalescing

Modeling the Problem

- Given: A minimal coloring of the IG
- Find a feasible coloring with minimal costs
 - Costs are a weighted sum over all equal-color edges
 - Unused colors may be used
 - Structure of graph and program must not be changed



Coalescing

Formal Statement

- Find a k -coloring \mathcal{C} of the IG which assigns as many ϕ -operands and results the same color.

$$\min_{\mathcal{C}} \sum_{\phi} \text{costs}(\mathcal{C}, \phi)$$

where

$$\text{costs}(\mathcal{C}, y \leftarrow \phi(x_1, \dots, x_n)) = \sum_{i=1}^n \begin{cases} 0 & \text{if } \mathcal{C}(y) = \mathcal{C}(x_i) \\ w_{yx_i} & \text{else} \end{cases}$$



Coalescing

Solution Strategies

Complexity

- Problem is NP-complete in number of ϕ s

Algorithms

- A greedy heuristic
- An optimal method using ILP (integer linear programming)



Coalescing

Heuristic

- Idea: Change colors to achieve more affine pairs
- Problem: Not locally decidable if color change is possible
- Therefore
 - Consider each ϕ separately
 - Try to give ϕ -operands and result the same color
 - Resolve color clashes recursively through the graph
 - On failure mark the conflict locally and repeat



Coalescing

Formalization as ILP

- Binary variables represent states/decisions
- Coloring: $x_{ic} = 1 \Leftrightarrow$ node i has color c
- Optimality: $y_{ij} = 1 \Leftrightarrow$ node i and j have different colors

$$\min f = \sum_{e \in Q} w_e \cdot y_{ij}$$

$$\text{where } \sum_c x_{ic} = 1 \quad v_i \in V$$

$$x_{ic} + x_{jc} \leq 1 \quad [v_i, v_j] \in E$$

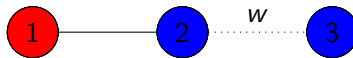
$$y_{ij} \geq x_{ic} - x_{jc} \quad [v_i, v_j] \in Q$$

$$y_{ij}, x_{ic} \in \{0, 1\}$$



Coalescing

ILP Example with 3 nodes and 3 colors



$$\min \quad ???w \cdot y_{23}$$

$$\text{where } \begin{array}{l} x_{11} + x_{12} + x_{13} = 1 \\ x_{21} + x_{22} + x_{23} = 1 \\ x_{21} + x_{22} + x_{23} = 1 \\ x_{21} + x_{22} + x_{23} = 1 \\ x_{31} + x_{32} + x_{33} = 1 \end{array} \quad \begin{array}{l} \text{coloring} \\ \text{coloring} \\ \text{coloring} \end{array}$$

$$\downarrow$$

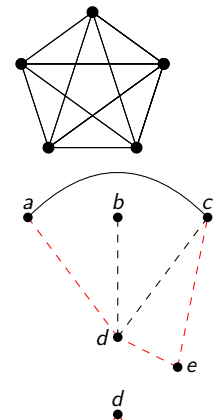
$$\begin{array}{l} x_{11} + x_{21} \leq 1 \\ x_{12} + x_{22} \leq 1 \\ x_{13} + x_{23} \leq 1 \\ x_{11} + x_{21} \leq 1 \\ x_{12} + x_{22} \leq 1 \\ x_{13} + x_{23} \leq 1 \end{array} \quad \begin{array}{l} \text{interference} \\ \text{interference} \end{array}$$



Coalescing

Improving the ILP Runtime

- Clique inequalities
- Path inequalities
- Clique-Path inequalities



Replace $O(n^2)$ inequalities $x_{ic} + x_{jc} \leq 1$ with one: $\sum_{i=1}^n x_{ic} \leq 1$

Use incompatibility of interference- and equal-color-edges
 $y_{ad} + y_{cd} \geq 1$
 $y_{ad} + y_{de} + y_{ec} \geq 1$

Multiple use of same argument,

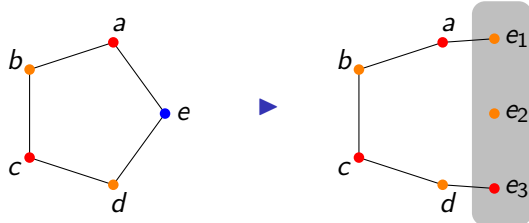
$$\text{where } \begin{pmatrix} a \\ b \end{pmatrix} = \Phi \begin{pmatrix} d & e \\ d & f \end{pmatrix}$$



Conclusions

SSA vs. non-SSA

- SSA-Construction introduces copies (ϕ -operations are copies along edges)
- These copies “blow up” the interference graph
- Nodes are replaced by stable sets



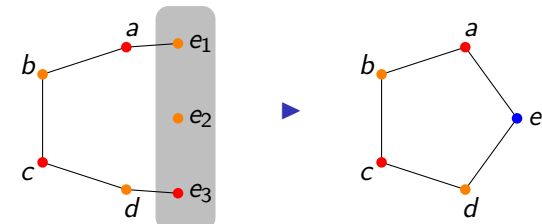
- Breaks cycles in the interference graph
- The interference graph becomes chordal



Conclusions

Non-SSA vs. SSA

- SSA-Destruction coalesces copies aggressively without considering the number of available registers
- Stable sets merged into nodes



- Possibly creating cycles
- Possibly increasing the chromatic number of the graph



Conclusions

- Chordality of SSA IGs allows for decoupling spilling and coalescing
- Register pressure is a precise measure for the number of registers needed
- We know in advance where values must be spilled
 - ▶ All labels where the pressure is larger than k
- Spilling can be done before coloring **and**
- coloring will always succeed afterwards
- Coalescing re-expressed by introducing a cost function on colorings and finding a preferably **good** coloring

Architecture without iteration

