

COMP 412 FALL 2010

Code Optimization, Part III Global Methods

Comp 412

Copyright 2010, Keith D. Cooper & Linda Torczon, all rights reserved.

Students enrolled in Comp 412 at Rice University have explicit permission to make copies of these materials for their personal use.

Faculty from other educational institutions may use these materials for nonprofit educational purposes, provided this copyright notice is preserved.

The Story So Far ...

Introduced scope of optimization

- Local a single basic block
- Regional a subset of the blocks in a procedure
- Global an entire procedure
- Whole Program multiple procedures

Some example optimizations

- Local Value Numbering
- Superlocal Value Numbering
- Loop Unrolling
- Data-analysis & an application
- Procedure Placement



Intraprocedural

Interprocedural

Finding Uninitialized Variables



3

We can use global data-flow analysis to find variables that might be used before they are ever defined

A variable v is live at point p iff \exists a path in the CFG from p to a use of v along which v is not redefined.

Any variable that is live in the entry block of a procedure may be used before it is defined

- Represents a potential logic error in the code
- Compiler should discover and report this condition

We are looking at this issue because it gives us an opportunity to introduce the computation of *liveness*

- Compiler builds a CFG and computes LIVEOUT(n) \forall node n
- LIVEOUT is a classic problem in data-flow analysis

Live Variables



Data-flow problems are expressed with a set of simultaneous equations over sets associated with nodes in a graph

 $LIVEOUT(n_f) = \emptyset$

 $LIVEOUT(n) = U_{m \in succ(n)} (UEVAR(m) \cup (LIVEOUT(m) \cap VARKILL(m)))$

Where

- UEVAR(n) is the set of names used before being defined in the block that corresponds to node n in the CFG
- VARKILL(n) is the set of names defined in the block that corresponds to node n in the CFG

These equations annotate each CFG node *n* with a LIVEOUT set

Live Variables



To compute live information for a procedure

- Build the CFG
- Compute UEVar and VarKILL sets
- Use an iterative fixed-point solver to compute LiveOut sets

 $N \leftarrow number of blocks$ for i = 0 to N-1 $LIVEOUT(i) \leftarrow \emptyset$ changed \leftarrow true while(changed) changed \leftarrow false for $i \leftarrow 0$ to N-1 recompute LIVEOUT(i) if LIVEOUT(i) changed then changed \leftarrow true

Iterative fixed-point solver

- LIVEOUT $\subseteq 2^{\text{Names}}$
- UEVAR, VARKILL are constants
- Equation is monotone increasing
- Finite sets + monotone equations
 ⇒ algorithm must halt

Theory of data-flow analysis assures us that this equation has a unique fixed point solution



A couple more points

- Can use LIVEOUT sets to find uninitialized variables
 - $-x \in LIVEOUT(n_0)$ means x is uninitialized at some use
- Can use LIVEOUT sets to eliminate unnecessary stores
 - Build LIVE at each operation
 - $-x \notin$ Live at a store means that the value is never reloaded
- The LIVEOUT equations have a unique fixed-point solution
 - ⇒ The algorithm finds a fixed-point solution; since the fixed-point solution is unique, it finds the correct solution
- Order of computation determines speed of convergence
 - ⇒ Choose an order that reaches fixed point quickly



The order of computation affects speed of convergence

- Live is a backward data-flow problem
 - Sets for node n are computed from sets at n's CFG successors



LIVEOUT(1) is computed from LIVEOUT(2) and LIVEOUT(3), which depend on LIVEOUT(4)

- Solver should visit 2 & 3 before 1
- Solver should visit 4 before 2 & 3
- Update as many "sources" as possible before visiting a given node

General idea is to let a change in the LiveOut set flow as far in the CFG as it can in a single "pass" of the while loop



Code optimization is intimately tied to graph theory

- LIVEOUT is computed, conceptually, on the reverse CFG
- Order for solver is defined by the reverse CFG



Propagation Order for Backward Problem

- Want to visit 4, then {2,3}, then 1
- PostOrder of RCFG would be 1, {2, 3}, 4
- Desired order is reverse of postorder
 - \rightarrow RPO(i) = |N| + 1 PO(i)
- Reverse PostOrder would be 4, {3, 2}, 1
 - \rightarrow We don't care about order of 2 & 3

```
Forward problem \Rightarrow RPO on the CFG
Backward problem \Rightarrow RPO on the reverse CFG
RPO on RFG is not reverse preorder on CFG
```



Define the problem

A variable v is live at point p iff \exists a path in the CFG from p to a use of v along which v is not redefined.

Solve the equations for LIVEOUT

 $LIVEOUT(n_f) = \emptyset$

 $\mathsf{LIVEOUT}(n) = U_{m \in \mathit{succ}(n)} (\mathsf{UEVar}(m) \cup (\mathsf{LIVEOUT}(m) \cap \overline{\mathsf{VarKILL}(m)}))$

Use an iterative fixed-point solver

- Theory guarantees unique solution to this problem
- Choose an order that produces efficient solution

Use the sets to identify uninitialized variables, or to eliminate useless stores, or to find live ranges, or to ...

EaC1e has material on Live in $~\S~$ 9.2

Chapter 9 of EaC contains more examples of data-flow problems