EECS 583 – Class 4 If-conversion

University of Michigan

September 19, 2011

Announcements & Reading Material

- Reminder HW 1 due Friday
 - » Talk to Daya this week if you are having troubles with LLVM
- Today's class
 - » "The Program Dependence Graph and Its Use in Optimization",
 - J. Ferrante, K. Ottenstein, and J. Warren, ACM TOPLAS, 1987
 - This is a long paper the part we care about is the control dependence stuff. The PDG is interesting and you should skim it over, but we will not talk about it now
 - » "On Predicated Execution", Park and Schlansker, HPL Technical Report, 1991.
- Material for Wednesday
 - » "Effective Compiler Support for Predicated Execution using the Hyperblock", S. Mahlke et al., MICRO-25, 1992.
 - » "Control CPR: A Branch Height Reduction Optimization for EPIC Processors", M. Schlansker et al., PLDI-99, 1999.

From Last Time: Predicated Execution

a = b + c
if (a > 0)
e = f + g
else
e = f / g
h = i - j

BB1	add a, b, c
BB1	bgt a, 0, L1
BB3	div e, f, g
BB3	jump L2
BB2	L1: add e, f, g
BB4	L2: sub h, i, j



Traditional branching code

p2 → BB2 p3 → BB3	BB1	add a, b, c if T	
	BB1	p2 = a > 0 if T	BB1
	BB1	p3 = a <= 0 if T	BB2
	BB3	div e, f, g if p3	BB3
	BB2	add e, f, g if p2	BB4
	BB4	sub h, i, j if T	

Predicated code

From Last Time: Class Problem



p1 = cmpp.UN(a > 0) if T p2, p3 = cmpp.UNUC(b > 0) if p1 r = t + s if p2 u = v + 1 if p3 y = x + 1 if p1

- a. Draw the CFG
- b. Predicate the code removing all branches

If-conversion

- Algorithm for generating predicated code
 - » Automate what we've been doing by hand
 - » Handle arbitrary complex graphs
 - But, acyclic subgraph only!!
 - Need a branch to get you back to the top of a loop
 - » Efficient
- Roots are from Vector computer days
 - » Vectorize a loop with an if-statement in the body
- ✤ 4 steps
 - » 1. Loop backedge coalescing
 - » 2. Control dependence analysis
 - » 3. Control flow substitution
 - » 4. CMPP compaction
- My version of Park & Schlansker

Running Example – Initial State



Step 1: Backedge Coalescing

- Recall Loop backedge is branch from inside the loop back to the loop header
- This step only applicable for a loop body
 - » If not a loop body \rightarrow skip this step
- Process
 - » Create a new basic block
 - New BB contains an unconditional branch to the loop header
 - » Adjust all other backedges to go to new BB rather than header
- Why do this?
 - » Heuristic step Not essential for correctness
 - If-conversion cannot remove backedges (only forward edges)
 - But this allows the control logic to figure out which backedge you take to be eliminated
 - » Generally this is a good thing to do

Running Example – Backedge Coalescing



Step 2: Control Dependence Analysis (CD)

- Control flow Execution transfer from 1 BB to another via a taken branch or fallthrough path
- Dependence Ordering constraint between 2 operations
 - » Must execute in proper order to achieve the correct result
 - » O1: a = b + c
 - » O2: d = a e
 - » O2 dependent on O1
- Control dependence One operation controls the execution of another
 - » O1: blt a, 0, SKIP
 - » O2: b = c + d
 - » SKIP:
 - » O2 control dependent on O1
- Control dependence analysis derives these dependences

Control Dependences

- Recall
 - » Post dominator BBX is post dominated by BBY if every path from BBX to EXIT contains BBY
 - » Immediate post dominator First breadth first successor of a block that is a post dominator
- Control dependence BBY is control dependent on BBX iff
 - There exists a directed path P from BBX to BBY with any BBZ in P (excluding BBX and BBY) post dominated by BBY
 - » 2. BBX is not post dominated by BBY
- In English,
 - » A BB is control dependent on the closest BB(s) that determine(s) its execution
 - » Its actually not a BB, it's a control flow edge coming out of a BB



Running Example – CDs



First, nuke backedge(s)
Second, nuke exit edges
Then, Add pseudo entry/exit nodes
 - Entry → nodes with no predecessors

- Exit \rightarrow nodes with no successors

Control deps (left is taken) BB1: BB2: BB3: BB3: BB4: BB5: BB5: BB6: BB7: BB7: BB8: BB8: BB9:

Algorithm for Control Dependence Analysis

```
for each basic block x in region
  for each outgoing control flow edge e of x
     y = destination basic block of e
     if (y not in pdom(x)) then
       lub = ipdom(x)
       if (e corresponds to a taken branch) then
          x id = -x.id
       else
                                                            Notes
          x id = x.id
       endif
                                            Compute cd(x) which contains those
       \mathbf{t} = \mathbf{y}
                                           BBs which x is control dependent on
       while (t != lub) do
          cd(t) += x_id;
                                              Iterate on per edge basis, adding
          t = ipdom(t)
                                            edge to each cd set it is a member of
       endwhile
     endif
   endfor
endfor
```

Running Example – Post Dominators



	<u>pdom</u>	ipdom
BB1 :	1, 9, ex	9
BB2:	2, 7, 8, 9, ex	7
BB3 :	3, 9, ex	9
BB4:	4, 7, 8, 9, ex	7
BB5 :	5, 7, 8, 9, ex	7
BB6 :	6, 7, 8, 9, ex	7
BB7 :	7, 8, 9, ex	8
BB8 :	8, 9, ex	9
BB9 :	9, ex	ex

Running Example – CDs Via Algorithm



edge (al	<u>(a -1)</u>
	x = 1
	$e = taken edge 1 \rightarrow 2$
	y = 2
	y not in pdom(x)
	lub = 9
	$x_id = -1$
	t = 2
	2 != 9
	cd(2) += -1
	t = 7
	7 != 9
	cd(7) += -1
	t = 8
	8 != 9
	cd(8) += -1
	t = 9
	9 == 9

Running Example – CDs Via Algorithm (2)



Running Example – CDs Via Algorithm (3)





Step 3: Control Flow Substitution

- Go from branching code \rightarrow sequential predicated code
- 5 baby steps
 - » 1. Create predicates
 - » 2. CMPP insertion
 - » 3. Guard operations
 - » 4. Remove branches
 - » 5. Initialize predicates

Predicate Creation

- R/K calculation Mapping predicates to blocks
 - » Paper more complicated than it really is
 - » K = unique sets of control dependences
 - » Create a new predicate for each element of K
 - » R(bb) = predicate that represents CD set for bb, ie the bb's assigned predicate (all ops in that bb guarded by R(bb))

K = { $\{-1\}, \{1\}, \{-2\}, \{-4\}, \{2,4\}, \{-1,-3\}\}$ predicates = p1, p2, p3, p4, p5, p6

CMPP Creation/Insertion

- For each control dependence set
 - » For each edge in the control dependence set
 - Identify branch condition that causes edge to be traversed
 - Create CMPP to compute corresponding branch condition
 - OR-type handles worst case
 - guard = True
 - destination = predicate assigned to that CD set
 - Insert at end of BB that is the source of the edge

K = { $\{-1\}, \{1\}, \{-2\}, \{-4\}, \{2,4\}, \{-1,-3\}$ } predicates = p1, p2, p3, p4, p5, p6

 $p1 = cmpp.ON (b < 0) if T \rightarrow BB1$

Running Example – CMPP Creation



Control Flow Substitution – The Rest

- Guard all operations in each bb by R(bb)
 - » Including the newly inserted CMPPs
- Nuke all the branches
 - » Except exit edges and backedges
- Initialize each predicate to 0 in first BB

bb = 1, 2, 3, 4, 5, 6, 7, 8, 9
CD(bb) = {{none},
$$\{-1\}, \{1\}, \{-2\}, \{-4\}, \{2,4\}, \{-1\}, \{-1,-3\}, \{none\}$$

R(bb) = T p1 p2 p3 p4 p5 p1 p6 T

Running Example – Control Flow Substitution



Loop:

```
p1 = p2 = p3 = p4 = p5 = p6 = 0
  b = load(a) if T
  p1 = cmpp.ON (b < 0) if T
  p2 = cmpp.ON (b \ge 0) if T
  p6 = cmpp.ON (b < 0) if T
  p3 = cmpp.ON (c > 0) if p1
  p5 = cmpp.ON (c <= 0) if p1
  p4 = cmpp.ON (b > 13) if p3
  p5 = cmpp.ON (b <= 13) if p3
  b = b + 1 if p4
  c = c + 1 if p5
  d = d + 1 if p1
  p6 = cmpp.ON (c <= 25) if p2
  e = e + 1 if p2
  a = a + 1 if p6
  bge e, 34, Done if p6
  jump Loop if T
Done:
```

Step 4: CMPP Compaction

- Convert ON CMPPs to UN
 - » All singly defined predicates don't need to be OR-type
 - » OR of 1 condition → Just compute it !!!
 - » Remove initialization (Unconditional don't require init)
- Reduce number of CMPPs
 - » Utilize 2nd destination slot
 - » Combine any 2 CMPPs with:
 - Same source operands
 - Same guarding predicate
 - Same or opposite compare conditions

Running Example - CMPP Compaction

```
Loop:
  p1 = p2 = p3 = p4 = p5 = p6 = 0
  b = load(a) if T
  p1 = cmpp.ON (b < 0) if T
  p2 = cmpp.ON (b \ge 0) if T
  p6 = cmpp.ON (b < 0) if T
  p3 = cmpp.ON (c > 0) if p1
  p5 = cmpp.ON (c \le 0) if p1
  p4 = cmpp.ON (b > 13) if p3
  p5 = cmpp.ON (b <= 13) if p3
  b = b + 1 if p4
  c = c + 1 if p5
  d = d + 1 if p1
  p6 = cmpp.ON (c <= 25) if p2
  e = e + 1 if p2
  a = a + 1 if p6
  bge e, 34, Done if p6
  jump Loop if T
Done:
```

Loop: p5 = p6 = 0b = load(a) if T p1,p2 = cmpp.UN.UC (b < 0) if Tp6 = cmpp.ON (b < 0) if Tp3,p5 = cmpp.UN.OC (c > 0) if p1p4,p5 = cmpp.UN.OC (b > 13) if p3b = b + 1 if p4 c = c + 1 if p5 d = d + 1 if p1 p6 = cmpp.ON (c <= 25) if p2 e = e + 1 if p2 a = a + 1 if p6 bge e, 34, Done if p6 jump Loop if T Done:

Class Problem

if
$$(a > 0) \{$$

 $r = t + s$
if $(b > 0 \parallel c > 0)$
 $u = v + 1$
else if $(d > 0)$
 $x = y + 1$
else
 $z = z + 1$
}

- a. Draw the CFG
- b. Compute CD
- c. If-convert the code

Next Time – When to If-convert, When to use Branches