CS202 Compiler Construction

March 27, 2003

Today:

Assignment 8 overview, code description

Introduction to instruction selection

Question: how many of you are taking this class for undergrad credit?

Assignment 8

Two written problems
Two functions to implement
IRFunction.findBlocks
IRFunction.traceBlocks
Each moderate sized
Many library functions provided
Do the written part first
Will give better conceptual understanding, independent of implementation details
Important new java class: BasicBlock

BasicBlock contains label, jump, list of statements

Maintains list of statements
  first is label
  last is jump (IRJump or IRCJump)

BBlist, BBiter also provided in distribution

BasicBlock maintains next, prev Bblists

Next list are those blocks that can be executed after this

Prev list are those blocks that can be executed immediately before this

Create block with just a label

Add statements incrementally
  Use BasicBlock.stmts().add method

Set jump specifically
  Use setJump method, calls addStmt
Call addNext for each label
  true/false for jump
  default and nthLabel for jump

IRCJump.trueLabel returns true label
IRCJump.falseLabel returns false label
IRJump.defaultLabel returns default label
IRJump.nthLabel(n) returns nth label in jump
IRJump.numberCases returns number of non-default cases in jump

addNext calls addPrev
  and vice versa

If a block b is next wrt b', then b' is prev wrt b

Need removeNext in building trace if changing target label
  Particular case of canonicalizing cjumps

IRFunction creates _blockIndex
  hashMap from label to block
  Maintains "known" basic blocks
You insert as each block created
  Use _blockIndex.put(lab, block)
  May create before encountered
Lookup to find block from label
  Use _blockIndex.get(lab)
  Must cast result as BasicBlock
If preexisting label, check whether block already exists before creation
For reordering false targets for jumps:

Need removeNext in building trace if changing target label

BasicBlock.jump returns jump statement
IRCJump.swapTrueFalse swaps true and false labels, negates condition
IRCJump.setTrue sets true label
IRCJump.setFalse sets false label

To obtain basic blocks from statement list
Start at beginning of list
Expect to see label
Add code to existing block until either new label encountered…
Close previous block
Append jump at end
or until jump/cjump encountered
Close this block

Canonicalizing cjumps, given BBlist:

while input BBlist list not empty
    remove first block b from list
    start new trace with b
    if b.jump is cjump:
        if target of cjump in list:
            choose false target if in list
            else if true in list, swap true/false
            make next Bblock in reordering,
            remove from input list
if neither cjump target in list
create new block:
reset false target to block
block starts with label
followed by jump to old target
add to end of trace
terminates trace

Try to turn jumps into traces also, by
reposition targets after jumps
Where possible

Why basic blocks?

*Control flow analysis*

Abstracts details of operations, focuses
on flow of control in programs

Most common benefit: optimizations

Example: loop optimizations

Want to turn code within a loop into single trace

Allows code to persist in cache
Reordering blocks:

start with first block

find target of its jump
and false target of its jump

L0: MOVE T2,0
MOVE T1,0
JUMP L3

L1: MOVE T2,0
MOVE T1,0
JUMP L3

L3: CJUMP T1,4,LT,L1,L4
L4: MOVE %i0,T2
JUMP L5

now need to start new thread

L0: MOVE T2,0
MOVE T1,0
JUMP L3

L1: MOVE T2,+(T2,T1)
JUMP L2

L3: CJUMP T1,4,LT,L1,L4
L4: MOVE %i0,T2
JUMP L5

and finally done

L0: MOVE T2,0
MOVE T1,0
JUMP L3

L1: MOVE T2,0
MOVE T1,0
JUMP L3

L3: CJUMP T1,4,LT,L1,L4
L4: MOVE %i0,T2
JUMP L5
Basic blocks form units in control flow analysis

Basic blocks form nodes in control flow graphs

Graphical representation allows myriad of graph algorithms to apply

(Much) more on this later

IR not machine instructions close, but not there yet

1. need machine specific instr
   MOVE, CALL are still generic

2. need flat operands
   registers or memory
   not tree structured expressions
Going from IR to instructions called instruction selection

Choose (select) machine instructions that implement the IR

(1) is generally easy; for example:
- CJUMP NEQ -> bne
- JUMP (basic) -> ba
- MOVE -> mov

But what about those operands?

Flatten into sequence of instructions still use temps, not real registers

s -> x = p2[i]
yields IR

becomes code

```plaintext
MOVE
  IRMEMORY
  BINEXP +
  TEMP 10
  CONST 8
  IRMEMORY
  BINEXP *
  TEMP 11
  BINEXP *
  TEMP 12
  CONST 4
```

```plaintext
mult t12,4,113
add t13,t11,t14
ld 0(t14),t15
stor t15,8(t10)
```
Instruction selection based on patterns for target machine
  each pattern yields 1+ instruction

Instruction selection consumes trees from bottom up
  yields instructions in correct order
  also returns reduced tree

---

simple patterns

```
binop
  +
  reg x
  reg y

mem
  temp x
```

```
add x,y,tN
  temp N
```

```
Id 0(tx),tN
  temp N
```

---

Each pattern covers part (or all) of an IR tree
  Multiple tree coverings possible

Want “best” patterns to cover tree
  Called tiling the tree
  “Best” patterns based on cost of tiling

Tile from the bottom up
Conventions: C = const, T = temp, N = new temp

---
MOVE

MEM +
t10 8

mult t12,4,t13
add t11,t13,t14
ld 0(t14),t15

ld 0(t14),t15

stor t15,8(t10)
Many trees have multiple tilings
each is legal
may have different cost

Goal is “cheapest” tiling
as measured by runtime cost

For risc processors
#intras approximates cost
#tiles approximate #intras
find fewest tiles

Use maximal munch algorithm
greedy algorithm

Start at top of tree
find largest tile possible
recurse over children

Apply tiles on way up
generates code from bottom up

3 tiles fit top of tree
simple move

MOVE
MEM
+  
  t10  8
  t11
  t12  4
MEM
+

simple store

MOVE
MEM
+  
  t10  8
  t11
  t12  4
MEM
+

store with constant offset

MOVE
MEM
+  
  t10  8
  t11
  t12  4
MEM
+
Any architecture offers many tiles
too expensive to check them all

Arrange tiles by top level node kind
select list of tiles using switch/case

List ordered by cost (mostly size)
select first that matches