Christoph Kessler with Mattias Eriksson

DF00100
Advanced Compiler Construction
VT1 / 2010

Exercise Set 2

Exercise 2.1

Given the following program fragment:

for i **from** 1 **to** N **do**
S_2: $A[i] \leftarrow B[i] + 1$
od
for i **from** 1 **to** N **do**
S_5: $C[i] \leftarrow A[i] \ / \ 2$
od
for i **from** 1 **to** N **do**
S_8: $D[i] \leftarrow 1/C[i+1]$
od

(a) Draw the data dependence graph. Identify loop-carried and loop-independent dependences.

(b) Is it safe to apply loop fusion to all the three loops? Why or why not?

(c) Apply loop fusion as far as possible. Show the resulting data dependence graph.

(d) Discuss strategies that enable loop fusion for loops that (slightly) differ in their lower or upper bounds.

Exercise 2.2

Consider the safety conditions for loop distribution (aka. loop fission). Do loop-carried anti-dependences or output dependences that contribute to a dependence cycle / SCC really constitute a prohibiting factor for loop distribution? Propose a (safe) transformation to get rid of loop-carried anti- and output dependences, and give an example where this enables loop distribution.

Exercise 2.3

Apply loop tiling to the following loop nest:

```
for (i=0; i<n; i++)
   for (j=0; j<m; j++)
      X[i][j] = Y[i][j];
```

Exercise 2.4

Given the following loop nest:

```
for (i=0; i<n; i++)
   for (j=0; j<m/2; j++)
      for (k=1; k<r; k++) {
S1:      B[i][j] = fsin(i*w*t) + fcos(j*w*t);
S2:      A[i][k-1][2*j] = A[i+1][k][2*j+1] + B[i][j];
      }
```

(a) Draw the data dependence graph and determine the direction vectors of loop-carried dependences.

(b) Which loop headers are interchangeable?

(c) Identify loop-invariant code and move it to a more appropriate place.

Exercise 2.5

Loop distribution involves computing the strongly connected components (SCCs) of the data dependence graph. Recapitulate Tarjan's algorithm for computing SCCs (take a look into the DFS slides linked from the course homepage or any good book on algorithms, such as [Cormen/Leiserson/Rivest 1990]). What is the time complexity for Tarjan's algorithm?

Exercise 2.6

Given the following tree grammar (from the lecture):

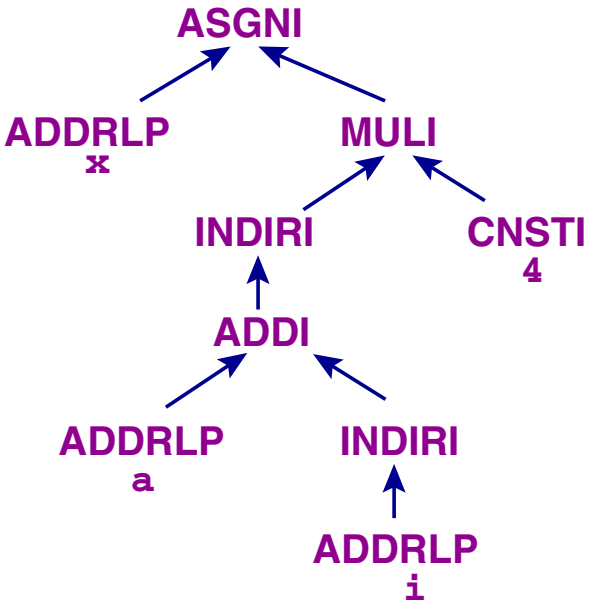
Nonterminals N = { stmt, reg, con, addr, mem, ... } where the start symbol is stmt;

terminals T = { CNSTI, ADDRLP, ... };

and the production rules with costs are as follows:

reg → ADDI(reg, con)	`addi %r,%c,%r`	1
reg → ADDI(reg, reg)	`addi %r,%r,%r`	1
reg → MULI(reg, con)	`muli %r,%c,%r; nop`	2
reg → MULI(reg, reg)	`muli %r,%r,%r; nop`	2
reg → MULI(reg, con) **if** con.value < 256 **and** isPowerOfTwo(con)	`lshi %r,log2(%c),%r`	1 (left shift)
stmt → ASGNI(addr, reg)	`store %r,%a`	1
stmt → ASGNI(reg, reg)	`store %r,0(%r)`	1
reg → ADDRLP	`addi fp,#%d,%r`	1
addr → ADDRLP	`%d(fp)`	0
reg → addr	`addi %a,%r`	1
reg → INDIRI(addr)	`load %a,%r; nop`	2
reg → INDIRI(reg)	`load 0(%r),%r; nop`	2
reg → INDIRC(addr)	`load %a,%r; nop`	2
reg → INDIRC(reg)	`load 0(%r),%r; nop`	2
reg → CVCI(INDIRC(addr))	`load %a,%r; nop`	2
reg → CVCI(INDIRC(reg))	`load 0(%r),%r; nop`	2
con → CNSTI	`%d`	0
reg → con	`addi R0,#%c,%r`	1

Use the dynamic programming algorithm of the lecture to find a least-cost derivation for the following LIR tree for `int x, i, a[];  x = a[i] * 4`:



Exercise 2.7

In the lecture on local instruction scheduling we described the list scheduling algorithm as a *forward scheduling* algorithm, that is, topological sorting in forward direction of the dependences.

Give the pseudocode of *backward* list scheduling.

Please prepare Exercises 2.1–2.5 for the second lesson, the remaining ones for the third lesson.