University of Cape Town

Department of Computer Science

CSC3003S Final Exam

2008

Marks : 30

Time : 3 hours

Instructions:

- Answer all questions from Section A and 3 questions from Section B.
- Show all calculations where applicable.

Section A [Answer Question ONE – this is compulsory]

Question 1 [10 marks]

Multi-core CPUs can influence the design of modern optimising compilers. Firstly, the compiler could automatically convert traditional serial code into a parallel form, where 2 or more pieces of code run at the same time on different CPUs/cores. Secondly, the compiler itself could perform its processing in parallel. Answer the following questions with this in mind.

1) Describe one advantage and one disadvantage of separating the compiler front-end from the compiler back-end. [2]

   ad: easier to retarget compiler to new machine [1], easier to apply optimisations to IR [1], easier to build compiler for new language [1]

   disad: more work [1] slower compilation [1]

2) Describe one advantage in having multiple distinct layers (frame generation, code generation, instruction selection, liveness analysis, etc.) within the compiler back-end. [1]

   easier to replace or change functionality in individual parts of compiler [1]

3) Static semantics should be checked before any code is generated. Discuss 2 examples of errors that can be checked for. [2]

   use before declaration [1] size mismatch [1] array index out of bounds [1] type incompatibility [1] etc.

4) Describe 3 typical optimisations that a compiler can perform on generated IR code, assuming the code being optimised runs on a single CPU/core. [3]

   trace analysis [1] common subexpression elimination [1] constant folding [1] etc.

5) At which point/layer in the compilation process could the code being compiled be converted into segments that may run in parallel (assuming this is done independently of source language)? [1]

after code generation but before liveness analysis [1]

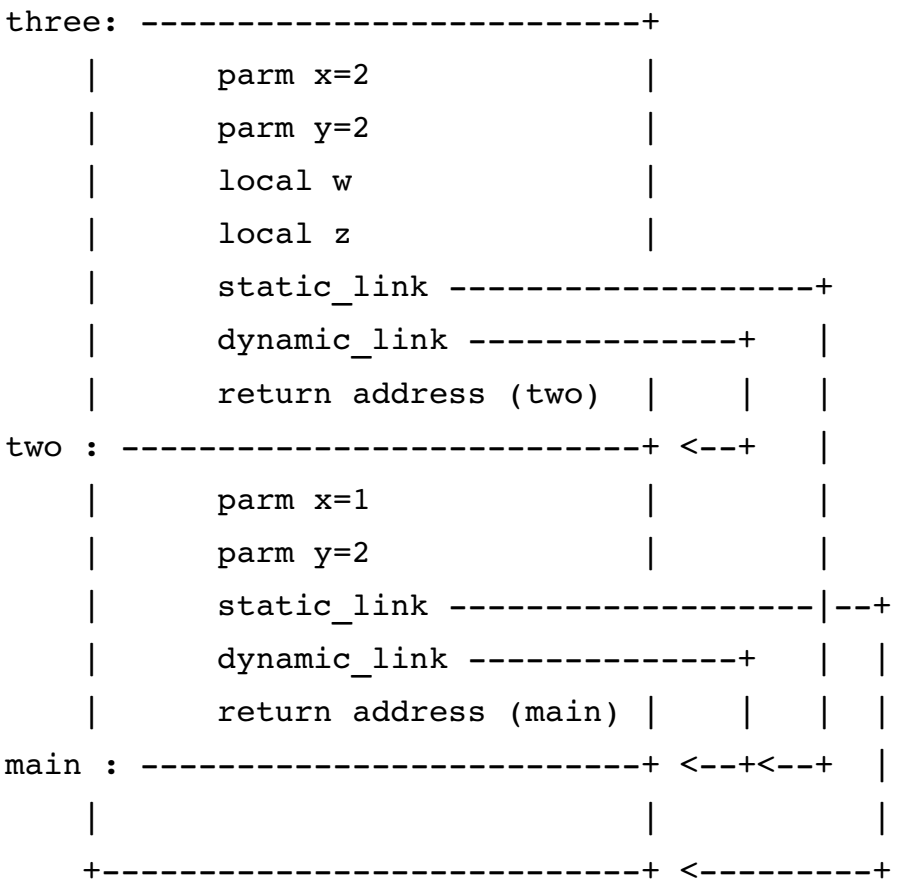
6) Briefly describe one technique to exploit multiple cores in the compilation process itself. [1]

after activation records, proceed in parallel [1] perform peephole optimisation in parallel [1] etc.

Section B [Answer 3 questions ONLY]

Question 5: Subprogram Analysis [10 marks]

a) Write the skeleton of a C-like program (with nested subprograms) that can result in the following activation record stack. All information that can be determined from the activation record stack must be indicated in the program. [5]



procedure main

[

call two(1, 2)

procedure two (x, y)

[

call three(2, 2)

procedure three (x, y)

[

var w, z

]

]

]

nesting [1] call sequence [1] locals [1] formal parameters [1] actual parameters [1]

b) For the following program, first separate the code into basic blocks, then rearrange the blocks into traces and finally optimise the code by removing redundant jumps. Show each step separately. [5]

```
Start:      Statement1
            Jump X
Z:          Statement2
            Jump A
X:          Statement3
            Jump Y
A:          Statement4
Y:          Statement5
            Jump Z
```

(a) breaking into blocks [1] adding jump [1]

```
Start:      Statement1
            Jump X
```

```
Z:          Statement2
            Jump A
```

```
X:          Statement3
            Jump Y
```

```
A:          Statement4
            Jump Y
```

```
Y:          Statement5
            Jump Z
```

(b) sequence of statements [1]

-

```
Start:      Statement1
            Jump X
X:          Statement3
            Jump Y
Y:          Statement5
```

Jump Z

Z: *Statement2*

Jump A

A: *Statement4*

Jump Y

(c) removal of redundant jumps [1] retaining of all labels and final jump [1]

Start: *Statement1*

X: *Statement3*

Y: *Statement5*

Z: *Statement2*

A: *Statement4*

Jump Y

Question 6: Register Allocation [10 marks]

a) Use the iterative liveness analysis algorithm to calculate the live-in and live-out sets for each of the following statements in a program, with the initial and final live sets indicated - assume live-in (succ (d=a+b+c)) = {d}. Use the statement numbers provided. Show the succ, use, def and pairs of in/out sets. The last calculated in/out pair must be identical to the previous pair to indicate convergence. [6]

```
[live-in: a, b]
1: if (a<b)
2:   then c = a;
3:   else c = b;
4: d = a + b + c;
[live-out: d]
```

Hint: The relevant formula can be stated as follows:

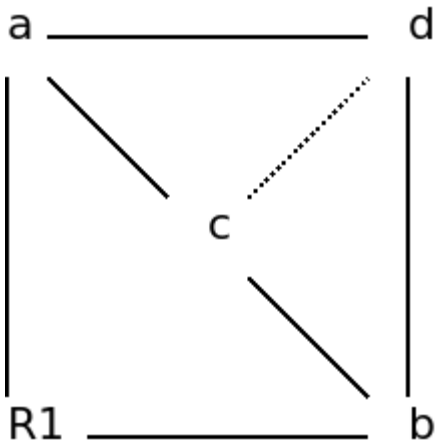
out[n] = union of in[s] for all successors s of n

in[n] = union of use[n] and (out[n] – def[n])

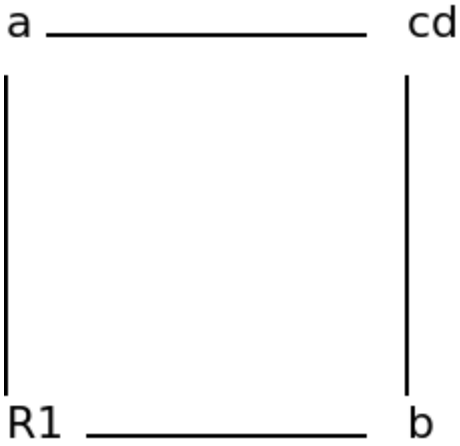
				Iteration 1		*Iteration 2*	
	Succ	*Use*	*Def*	*In*	*Out*	*In*	*Out*
In:a,b							
If (a<b)	*2,3*	*ab*	*-*	*ab*	*ab*	*ab*	*ab*
Then c = a	*4*	*a*	*c*	*ab*	*abc*	*ab*	*abc*
Else c = b	*4*	*b*	*c*	*ab*	*abc*	*ab*	*abc*
d = a+b+c	*-*	*abc*	*d*	*abc*	*d*	*abc*	*d*
Out: d							

Succ [1], Use [1], Def [1], In [1], Out [1], last two iterations being equal [1]

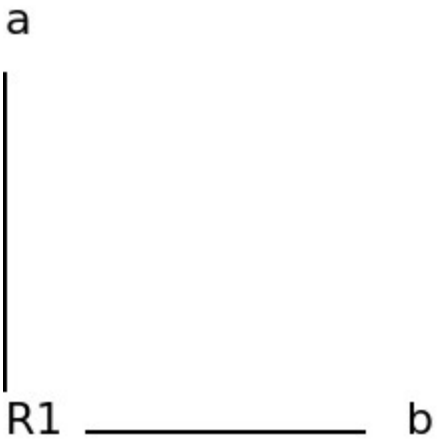
2. Consider the following graph with nodes indicating temporaries and arcs indicating interference. Apply a register colouring algorithm to 2-colour the graph. Assume that R1 is a precoloured node and use George's criterion for conservative coalescing. At each step, draw the updated graph and briefly explain what rule has been applied. Indicate the final register allocation (R1, R2) to temporaries. [4]



No nodes can be simplified, but c and d can be coalesced since each significant degree neighbour of c interferes with d. [1]



No nodes can be simplified or coalesced – choose cd as a potential spill and push onto stack. [1]



This makes a and b of <K degree, so they can be simplified. [1]

R1

Popping the nodes off the stack, we can then assign: [1]

A: R2

B: R2

C and D: R1