CONFIDENTIAL

Paper 3.32 MEng3 Computing Parallel Architectures

Department of Computing
Imperial College

Examination paper **March 27, 1998**

First examiner: . . . . **Second examiner: . . . .**

Course 332(MEng3 test) Parallel Architectures

This examination is of TWO HOURS' duration.

Answer ALL THREE questions.

1 Consider the following example code sequence:

```
if (a == 2)
  a = 0;
if (b == 2)
  b = 0;
if (a != b)
  c = 0
```

A straightforward implementation for the DLX machine might be as follows (`a` in `R1`, `b` in `R2`, `c` in `R3`, note that `R0` always contains zero):

```
SUBI  R4,R1,#2
BNEZ  R4,L1
ADD   R1,R0,R0   ; set R1 (a) to zero
L1:
SUBI  R4,R2,#2
BNEZ  R4,L2
ADD   R2,R0,R0   ; set R2 (b) to zero
L2:
SUBI  R4,R1,R2
BEQZ  R4,L3
ADD   R3,R0,R0   ; set R3 (c) to zero
```

a Modify the DLX code above to use delayed branches, where the instruction after the branch is always executed. If you can, put useful instructions in the delay slots.

b Use nullifying/cancelling delayed branches (`BNEZL` for branch likely, `BNEZU` for branch unlikely) to improve your implementation when `a` and `b` are usually expected to hold the value 2.

c Explain very briefly how the 1-bit branch prediction scheme could be used with a branch target buffer to improve the performance of the original DLX code given at the start of the question.

d Describe briefly how better branch prediction could be achieved for the third branch in the program (BEQZ R4,L3).

e Suppose the instruction set is extended with a conditional register-register move,

```
CMOVZ  R1,R2,R3  ; move R2 to R1 if R3 = 0
CMOVNZ R1,R2,R3  ; move R2 to R1 if R3 != 0
```

Show how this can be used in the example above. Does it yield good performance? What characterises the circumstances when it might be useful?

(The five parts carry, respectively, 20%, 15%, 25%, 15% and 25% of the marks).

2a Two machines are connected by together by a 3 metre link with a link bandwidth of 175 MB/s. The packet format consists of 10 bytes of header information and and 6 bytes of trailer information. The sender overhead is 100 μs and the receiver overhead is 120 μs. Calculate the effective bandwidth of the network when sending a payload of 128 bytes.

How would this vary for larger and smaller payloads? Given the small cache block sizes of shared memory architectures what aspect of the network design must be modified to maintain a high effective bandwidth?

b Many scientific codes involve transposing matrices. Assume that a $n \times n$ array representing the matrix is evenly partitioned onto the processors of a parallel machine. For a $p \times p$ mesh architecture the communication required by a transpose operation is given by:

processor (i, j) sends its partition of the array to processor (j, i)

Assume a 16 processor machine configured as a 4×4 mesh. The array is 1600×1600 elements large, where each element is a 4 byte float. The channels in the network have a bandwidth of 250 MB/s bidirectional. The time taken for the routing information to cross a switch is 0.25 μs. Assume XY worm-hole routing with virtual channels for flow control (you may assume an infinite number of virtual channels). The packet format has 10 bytes of header information and 6 bytes of trailer information. The payload is the size of the message. Assume that the sender overhead is 25 μs and the receiver overhead is 27 μs.

Draw the communication pattern taken by the sends in the algorithm

What is the time taken to complete the operation? (You may assume that the bandwidth of all the channels is that of the most heavily loaded channel.)

c In a shared memory architecture the algorithm is more likely to be:

processor (i, j) reads the partition of the array held by processor (j, i)

Assume a cache block size of 32 bytes.

What difference will supporting a shared memory architecture make?

How might an architecture designer hope to compensate for this difference?

(The three parts carry, respectively, 30%, 40% and 30% of the marks).

Turn over . . .

3 Consider the following program fragment:

```
declare A[0:N+1,0:M+1]
for i = 1 to N
  for j = 1 to M
S:  A[i,j] := A[i,j+1] + A[i,j-1]
```

a List the dependences present in this loop, indicating each dependence's type and its direction vector.

b For this part of the question, assume *(read these assumptions carefully!)*

- a 7-way dynamically-scheduled superscalar processor with register renaming and speculative execution using a re-order buffer
- no issue restrictions occur, i.e. that there are always enough functional units of the kind needed
- floating point operations are fully pipelined with a 3-cycle latency
- Loads take two cycles and stores take one cycle

Consider the following implementation of the *inner* loop above in DLX assembler:

```
     LD   R4, ...   !  address of A[i,1]
     LD   R14, ...  !  address of A[i,M+1]
  loop2:
S1:  LD   F0,-8(R4) !  load A[i,j-1]
S2:  LD   F1,+8(R4) !  load A[i,j+1]
S3:  ADDD F2,F0,F1
S4:  SD   0(R4),F2
S5:  ADDI R4,#8
S6:  SUBI R5,R4,R14 !  compare pointer to upper limit
S7:  BNEZ R5,loop2
```

Estimate the average number of clock cycles per iteration for this loop, assuming `M` is very large. You are not expected to produce a detailed timing diagram, but you should explain any stalls you anticipate. If you need to make further assumptions, state them clearly.

c Show using a diagram that the loops in the example given at the start of the question can be interchanged without changing the program's behaviour.

d For this part of the question, assume the same machine as above, although bear in mind that the data cache has a capacity of 64K bytes, with 64-byte (i.e. 8 word) blocks. Cache misses take 9 cycles, while cache hits take one.

Sketch how you would modify the source code of the program to achieve better performance.

(The five parts carry, respectively, 25%, 40%, 10% and 25% of the marks).

End of Paper