UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1998

BEng Honours Degree in Computing Part III
BEng Honours Degree in Information Systems Engineering Part III
MEng Honours Degree in Information Systems Engineering Part III
BSc Honours Degree in Mathematics and Computer Science Part III
MSci Honours Degree in Mathematics and Computer Science Part III
MEng Honours Degree in Electrical and Electronic Engineering Part IV
MSc Degree in Advanced Computing
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the
Diploma of Membership of Imperial College
Associateship of the City and Guilds of London Institute
Associateship of the Royal College of Science

PAPER 3.32 / I3.20 / E4.31

PARALLEL ARCHITECTURES

Tuesday, April 28th 1998, 10.00 - 12.00

Answer THREE questions

For admin. only: paper contains 4 questions

1a In a coherent-cache shared-memory multiprocessor, cache coherency can be maintained using either *update* or *invalidate* messages. Explain carefully why invalidation is more common.

b Consider the two following processes, running in parallel on a shared-memory multiprocessor:

```
/* processor 1 */
A := 0;
⋮
A := 1;
if (B == 0) {
  P();
}
```

```
/* processor 2 */
B := 0;
⋮
B := 1;
if (A == 0) {
  P();
}
```

Assume that procedure P is not called from elsewhere.

i) Assume that the processes are running on an idealised shared-memory machine. Can both processors execute P simultaneously? Explain your answer carefully.

ii) Assume that the two fragments are running on different CPUs in a coherent-cache shared-memory multiprocessor, using an invalidation protocol.
Can both processors execute P simultaneously? Explain the potential problem carefully.
What precautions are needed to preserve the expected behaviour?

iii) Some multiprocessors do not guarantee the expected behaviour for this example, but instead guarantee consistency only when CPUs synchronise (e.g. after passing through a LOCK or BARRIER). Explain why this might offer a performance advantage.

c Consider an 8-CPU coherent-cache shared-memory multiprocessor, using an invalidation protocol, in which directories (which indicate which caches hold a copy of a particular value) are represented using a singly-linked sharing list. Suppose the CPUs are interconnected using a full crossbar network.

i) What is the minimum number of messages needed to perform an invalidation? Explain your answer carefully.

ii) What is the maximum (i.e. worst-case) number of messages needed to perform an invalidation? Explain your answer carefully, using a diagram if necessary.

(The six parts carry, respectively, 20%, 15%, 15%, 10%, 10% and 30% of the marks).

2 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 dependencies present in this loop, indicating each dependency's type and its direction vector.

b For this part of the question, assume *(read these assumptions carefully; they are intended to simplify the question)*:

- a dynamically-scheduled superscalar processor with register renaming and speculative execution using a re-order buffer
- seven instructions can be issued per cycle
- 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 32K 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).

Turn over ...

3 Consider a parallel computer consisting of six processors connected in a ring, with bidirectional links between neighbours.

The link bandwidth is 1 Gigabit/second (10^9 bits/second), the switch delay 10ns, and the sender and receiver overheads are 1μs.

a Draw the channel dependency graph for this interconnection network.

b Explain, using your channel dependency graph, how deadlock could occur.

c Explain very briefly how virtual channels are implemented.

d Explain, using a channel dependency graph, how virtual channels can be used to overcome the deadlock problem in this machine.

e Suppose the processors are labelled $0 \ldots 5$, and every processor i sends a 1KByte message to processor $(i + 2) \bmod 6$.

Estimate the time for all messages to be delivered.

(The five parts carry, respectively, 20%, 15%, 20%, 15% and 30% of the marks).

4 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 to improve the performance of the original DLX code given at the start of the question. What additional hardware is needed?

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 at the start of the question. 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).

End of Paper