UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1996

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
MEng Honours Degree in Electrical and Electronic Engineering Part IV
MSc Degree in Foundations of Advanced Information Technology
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

Thursday, May 2nd 1996, 10.00 - 12.00

Answer THREE questions

For admin. only: paper contains
4 questions
4 pages (excluding cover page)

1 This question concerns the implementation of the following loop on a shared-memory multiprocessor:

```
for i1 = 1 to N do
 for j1 = 1 to N do
  for i2 = 1 to N do
   for j2 = 1 to N do
    if (A[i1,j1] > 0) and (A[i2,j2] > 0)
    then
     m = classify(100*((j2-j1)/(i2-i1)); /* (nearest int within bounds of 'line') */
     c = classify(100*(i1 - m*j1));      /* (nearest int within bounds of 'line') */
     line[m,c] = line[m,c] + 1;
    endif
   enddo
  enddo
 enddo
enddo
```

Assume that the number of processing elements is small compared to **N**, and that an invalidation-based cache coherence protocol is used.

a Is loop interchange valid here? Explain.

b What synchronisation issues arise in implementing this loop on a shared-memory multiprocessor?

c How can load imbalance be avoided without excessive management overheads?

d What aspects of the memory/cache architecture might influence the performance of a parallel implementation of this program on a coherent-cache shared-memory multiprocessor? Consider each of the memory references in the program in turn.

e *Briefly sketch* how this loop might be implemented efficiently on a coherent-cache shared-memory multiprocessor.

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

2 Consider the following loop:

```
for i = 1 to N do
S1: D[i] = E[i] * B[i+1]
S2: E[i] = D[i] * B[i+1]
enddo
```

a Draw a diagram to show the dependence structure for this loop. Indicate which array references are responsible for each dependence.

b Write down a vectorised version of this loop using a suitable high-level notation (such as Fortran 90).

c Assuming `N` is a multiple of 64, *sketch* how this loop would be implemented on a vector pipeline machine (such as a CRAY-1 or DLXV) with 64-word vector registers.

d Sketch a graph showing how the execution time would vary for this loop for increasing `N`. Explain your graph with reference to the architecture of a vector pipeline architecture.

e Sketch very briefly how the loop could be implemented on a coherent-cache shared-memory multiprocessor. What would influence its performance?

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

Turn over ...

3 This question concerns the performance of the following loop on two advanced static pipeline implementations of DLX:

```
for i = 1 to 100 do
  A[i] = B[i] + S * B[i-1]
enddo
```

Consider the following implementation for the DLX machine:

```
// Use R1 as ptr to B, R2 as ptr to A,
//     R3 as iteration count, initially 100, and
//     F0 as S
loop:   LD    F4, -8(R1)
        LD    F2, 0(R1)
        SUB   R3, R3, #1
        MULD  F4, F4, F0
        ADDI  R1, R1, #8
        ADDI  R2, R2, #8
        ADDD  F2, F2, F4
        SD    F2, -8(R2)
        BNEQZ R3, loop
```

a For this part of the question, assume a static pipelined implementation of the DLX processor with one non-pipelined, 2-cycle floating-point add/subtract unit, one non-pipelined, 4-cycle floating-point multiply unit, and an eight stage integer pipeline as follows:

IF	IS	RF	EX	DF	DS	TC	WB
Initiate I-cache access	Complete I-cache access	Decode instr'n and fetch registers	Execution	Initiate fetch from D-cache	Complete data fetch, and forward result	Tag check: check for cache hit	Write results to registers

(The MIPS R4000 processor has such a structure; the idea is called "superpipelining", and allows the clock period to be smaller than the cache access time).

Draw a diagram showing the execution of one iteration of this loop on this machine, assuming no cache misses, and assuming no stalls due to control hazards. Explain carefully where any forwarding or stalls are required. Explain any assumptions you have to make.

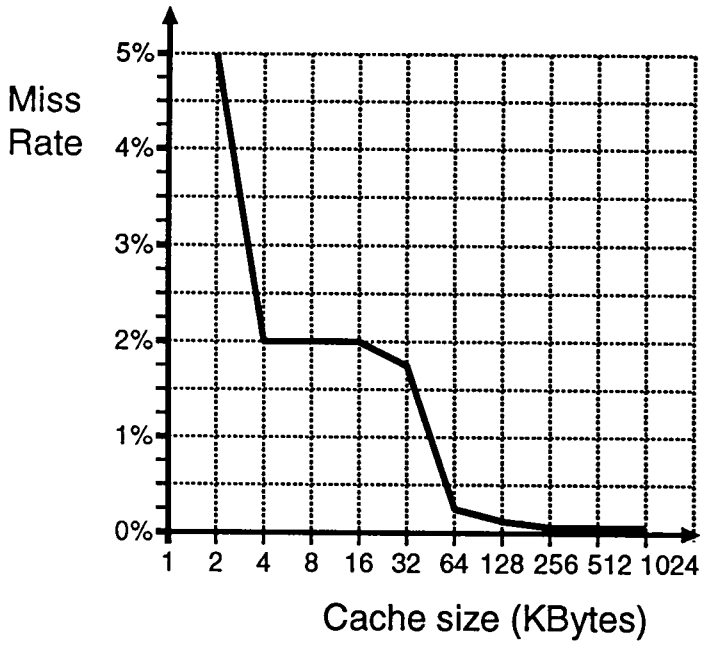
b For this part of the question, assume a similar pipeline, but modified so that each stage can potentially handle a pair of adjacent instructions. In such a "superscalar" design, two adjacent instructions can be issued per clock cycle provided they have no dependence, and the first updates only floating-point registers, while the second updates only integer registers. If these conditions cannot be met, the instructions are issued one at a time as in part (a).

(The DEC Alpha is an example of such a superscalar, superpipelined design).

Modify the assembly code given above to avoid as many stalls as possible. Explain carefully what transformations you apply, and why (**HINT:** consider more than one iteration).

(The two parts carry, respectively, 50% and 50% of the marks).

4 The graph below shows the approximate cache miss rate for a certain application, for increasing cache size:



Suppose the job is running on *one* processor of a two-processor shared-memory machine which uses a straightforward bus-based invalidation cache coherency protocol, with a cache for each processor, and a cache line size of 16 bytes. Assume the following:

1 25% of instructions are loads or stores.
2 The CPI assuming no cache misses is 1.5
3 The clock rate is 300MHz (clock period 3.33ns)
4 A cache miss takes 40 cycles (133ns)

a Calculate the MIPS rate of this application assuming no cache misses.

b Explain *two different* ways by which it might be possible to improve performance while reducing the MIPS rate.

c Estimate the CPI the system will achieve on this application for cache sizes of 4KB, 32KB and 64KB.

d Suppose that the operating system's scheduler decides to stop the job, and restart it on the other CPU. To do this involves a management overhead of 40μs (to save the registers of the process, and to restart the job on the new CPU by setting up the registers).
Suppose that the new CPU's cache contains no useful data, so the restarted job will suffer cache misses which would not have occured if the job had not been migrated.
Estimate the total delay attributable to additional cache misses each time the job is migrated, for the three cases when cache size is 4KB, 16KB and 64KB.

(The four parts carry, respectively, 10%, 20%, 30% and 40% of the marks).

End of Paper