UNIVERSITY OF LONDON

IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1999

BEng Honours Degree in Computing 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 Information Systems Engineering Part IV
MSc Degree in Advanced Computing
MSc Degree in Computing Science
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 3.25 / I 4.22

PARALLEL PROBLEM SOLVING

Wednesday, April 28th 1999, 2.30 – 4.30

Answer THREE questions

For admin. only:
paper contains 4 questions

1a Amdahl's law states that if the sequential component of a program accounts for the fraction *s* of the program's execution time, then the upper bound on the speedup of the program is *1/s*.

i) Derive the speedup formula for a general parallel program executing on P processors in terms of the fraction *s*, and show how the formula conforms to Amdahl's law for a large number of processors.

ii) Explain briefly the implications of Amdahl's law on exploiting parallelism within applications.

b i) Derive a simple performance model formula describing the execution behaviour of a generic load-balanced PIPE skeleton (i.e. co-ordination form). The model should account for all of the following:

1. the number of pipeline stages, n_p,
2. the computation time of each stage, t_f,
3. the time required to transfer data between processors, t_c,
4. the number of data elements passing through the pipeline, N.

(assuming that the number of processors, *P*, of a given machine is equal to n_p).

ii) What is the speedup formula for large values of N ?

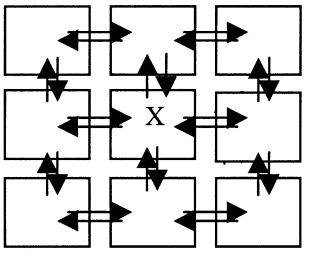
iii) The communication time required to transmit a message of size D bytes between two processors can be given as $t_c = t_0 + t_1 * D$. Usually, the value of t_0 (setting up time of communication) dominates the message transmission time. Suggest an optimisation method that can overcome such an effect on the performance of the PIPE skeleton when the value of N is large.

c Extend the performance model of the PIPE skeleton to cover the case when *P*, the number of processors of a given machine, is less than the number of pipeline stages (assume n_p is a multiple of *P*).

The three parts carry, respectively, 30% (15% for each subpart), 50% (20% , 10% and 20% for the three subparts respectively) and 20% of the marks.

© University of London 1999

2a A **2D_LNO** (two dimensional local neighbourhood operator) skeleton abstracts the behaviour of a wide class of parallel programs. Such programs operate by decomposing a 2D data structure on to a mesh-connected processor network. As shown in the figure below, in one iteration of the computation each processor operates on a local partition of the global data structure, and exchanges its boundary data with four neighbouring processors.



Derive a generic performance model for one iteration of the **2D_LNO** skeleton in terms of the models of the local computation and communication performed. To derive the model concentrate on the activities of the processor labelled "X".

b i) Describe the Jacobi and the Gauss-Seidel methods for solving the following partial differential equation:

$$x_{ij} = \frac{x_{i-1j} + x_{i+1j} + x_{ij+1} + x_{ij-1}}{4}$$

ii) Discuss the data dependency between iterations in the two methods and their impacts on strategies for parallelising the two methods.

c The following HPF program implements a data parallel version of the iterative Jacobi solver for the above partial difference equation

```
!HPF PROCESSORS pr( 2, 2)
real X(1000, 100). New(100, 100)
real err, err_tol
!HPF ALIGN New(: , : ) WITH X(:, : )
!HPF DISTRIBUTE X(BLOCK, BLOCK) ONTO pr
New (: , : ) = X ( : , : )
err= err_tol +1
DO WHILE (err > err_tol)
        X ( : , : ) = New (: , : )
        Forall (I=2:99, j=2:99)
        New ( i ,j ) = 0.25 *(  X(i-1, j ) + X( i+1, j) +X( i, j-1) +X( i, j+1) )
        Err  = MAXVAL(ABS(New-X))
END DO
```

Describe a SCL implementation of the data distribution, sharing and updating of the boundary information (**2D_LNO** operator) as indicated by this program.

The three parts carry, respectively, 30% , 35% (15% and 20% for two subparts respectively) and 35% of the marks.

Turn Over

© University of London 1999

3a i) Describe how Monte Carlo simulation can be used to compute an integral of the form :

$$\int_{-\infty}^{\infty} f(x)\,p(x)dx$$

where f(x) is some function and p(x) is a probability density.

ii) Describe a strategy for parallelising Monte Carlo simulation using the MPI **reduce** operator.

b Describe methods for implementing the **reduce** operator in a shared memory architecture and in a distributed memory architecture respectively.

c Write a SCL program fragment for data-parallel implementations of the following two linear algebra routines, based on a suitable data distribution strategy of your choice (you may assume that any necessary sequential program is already defined):

i) scalar vector product: which multiplies every element of a vector A by a scalar value s.

$$C_i = s * A_i$$

ii) dot product : where the result of the operation on two vectors A and B of the same length n is given by:

$$c = \sum_{i=1}^{n} A_i B_i$$

d Derive a suitable performance model formulae for your data-parallel implementations of each of the above two routines.

The four parts carry, respectively, 30% (15% for each subpart), 20%, 30% (15% for each subpart) and 20% of the marks.

© University of London 1999

4 The **DC** (divide-and conquer) skeleton abstracts the behaviour of a wide class of sequential and parallel algorithms. The general structure of such algorithms can be abstracted as follows:

```
DC  trivial solve divide combine x
= If  trivial x   then solve x
else
(combine o map (DC trivial solve divide combine) o  divide) x
```

where the function *trivial* tests if a problem is simple enough to be solved directly, *solve* solves the problem in this case, *divide* splits the problem into a list of sub-problems that can be solved recursively and the function *combine* joins the generated sub-solutions together.

a Outline briefly the two main strategies that can be used in parallelising **DC** algorithms. Indicate the conditions when each of these methods would be more appropriate.

b Use the above given higher-order definition of the **DC** skeleton to express the Quicksort algorithm.

c i) Describe how a task parallel implementation of Quicksort would work.

ii) In such an implementation discuss how the degree of parallelism changes as the computation proceeds and how does this change affects the overall achievable speedup of the algorithm.

iii) Derive an upper bound on the data to be communicated along the critical path of the execution of the algorithm when sorting an input list of length N on a distributed memory machine.

The three parts carry, respectively, 30%, 20% and 50 % (15% , 15% and 20% for the three subparts respectively) of the marks.

End of paper

© University of London 1999