UNIVERSITY OF LONDON
IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 1997

BEng Honours Degree in Computing Part III
MEng Honours Degree in Information Systems Engineering Part IV
BSc Honours Degree in Mathematics and Computer Science Part III
MSci Honours Degree in Mathematics and Computer Science Part III
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 / I4.22

PARALLEL PROBLEM SOLVING

Thursday, May 1st 1997, 10.00 - 12.00

Answer THREE questions

For admin. only: paper contains 4 questions

1a Discuss the concepts of *abstraction, efficiency and conventionality* with regard to parallel programming and explain how these concerns are addressed in the SCL co-ordination language.

b The following parallel C program with MPI approximates π on an n-processor distributed memory parallel computer:

```
#include "mpi.h"
#include <math.h>
double f(a)
double a;
{ return (4.0 / (1.0 + a*a));
}
int main(argc,argv)
int argc;
char *argv[];
{
        int done = 0, n, myid, numprocs, i, rc;
        double PI25DT = 3.141592653589793238462643;
        double mypi, pi, h, sum, x, a;
        double startwtime, endwtime;
        MPI_Init(&argc,&argv);
        MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
        MPI_Comm_rank(MPI_COMM_WORLD,&myid);
        n =100;

        MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD ) ;
        sum = 0.0;
        for (i = myid + 1; i <= n; i += numprocs)
        {
                x = h * ((double)i - 0.5);
                  sum += f(x);
        }
         mypi = h * sum;
        MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0,
        MPI_COMM_WORLD);
        if (myid == 0)
        {
                printf("pi is approximately %.16f, Error is %.16f\n",
                pi, fabs(pi - PI25DT));
          }
         MPI_Finalize();
}
```

Discuss the parallel behaviour of this program and explain the functionality of each MPI construct used.

c Use the SCL Partition and SPMD co-ordination forms to specify the above program.

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

2a Describe how the data partition operation of SCL abstracts the notion of data physically distributed around a parallel machine and discuss how SCL co-ordination forms can be used for efficiently distributing sparse matrices.

b) The co-ordination structure of the Jacobi code for solving the Poisson equation is as follows:

```
Jacobi ips n X =
   let
       (dx+ br, f ) = partition row_colu_block (n + (1, 1) ) ( n+ (1, 1) ) X
   in  IterUntil conv finalsol step <dx+br, ips>
              final sol <dx+ br, dconv > = gather dx
              conv <dx+ br, dconv > = (fold max dconv ) < ips
              step = SPMD [ (gf, Localsolver) ]
       where
              gf < dx+ br, dconv > = < dx + (update f dx) , dconv>
```

Localsolver is a provided sequential Jacobi solver.

Discuss the requirements for data sharing in the above program and explain how these are supported by the SCL constructs used. In particular what is the role of the function f?

c) Suppose we wish to map an N*N mesh onto P processors. Discuss the performance model of above program by examining the communication and computation costs of each iteration. (To simplify the analysis assume that N is a multiple of $\sqrt{P}$.).

The three parts carry, respectively, 40%, 40% and 20% of the marks.

Turn over ....

3a Discuss the main difficulties involved in programming irregular and adaptive data parallel applications and the principles involved in designing parallel data types in order to solve problems of this type.

b) The Tree parallel abstract data type is designed to support the programming of parallel adaptive computations. Suppose the data type has abstract constructors Node and Leaf and provides the following four operators:

1) ***New_Tree*** *f m* : builds up a tree from the data set m according to the function f.

2) ***Map_Leaf*** *f T* : maps a function f to all leaves of the tree T

3) ***Tree_Transfer*** *leaff nodef T* : rebuilds the tree *T* by applying the function *leaff* to all the leaves and the function *nodef* to all the nodes of the tree using post order traversal.

4) ***Tree_Fold*** *con final rf T* : recursively reduces the tree according to the reduction operation *rf* if the condition *con* is met; otherwise applies the function *final* to the whole tree.

Thus, for example, ***Map_Leaf*** could have the following functional definition

Map_Leaf f (Leaf l) = Leaf (f l)

Map_Leaf f (Node v [T_1, T_2, ..,T_N])
= Node v [(Map_Leaf f T_1), (Map_Leaf f T_2), .., (Map_Leaf f T_N)]

Write similar functional definitions for the operators ***Tree_Transfer, Tree_Fold*** and discuss their possible parallel behaviour.

c) Use the Tree parallel data type to code the overall structure of the Barnes-Hut algorithm for particle simulation. You may assume that any necessary sequential program, such as one for computing the gravitational function, is already defined.

The three parts carry, respectively, 25%, 45%, and 30% of the marks.

4a Outline the main differences between data parallelism and task parallelism and discuss the issues involved in building complex parallel applications from individual data- or task-parallel components.

b Given a data set S and a set of class values C the following procedure implements the construction of a decision tree in a classification algorithm. Parallelise the procedure using the DC (divide and conquer) co-ordination form and discuss the advantages and disadvantages of your parallelisation.

```
Tree_Construction (C, S)
      let A = class(S) in
            if A  = {c}then Leaf c  /i.e. class(S) is a singleton/
            else
                  let  a = Select_attribute(S, C)
                  [S1, ..,Sn ] = Test (a, S)
                  in  Node a [ Ti | Ti = Tree_Construction (C, Si ) ]
where :
class (S): returns the class values of the elements in S
Select_attribute(S, C): selects an attribute for testing
Test(a, S):  partitions the data set S based on a pre-defined test
```

c Outline a data parallel version of the algorithm discussed in b.

The three parts carry, respectively, 35%, 45%, 20% of the marks.

End of Paper