CS4021

THE UNIVERSITY OF WARWICK
MSc Examinations: Summer 2018
High Performance Computing

Time allowed: 2 hours
Choose to answer TWO questions from Questions 1, 2 and 3.
Question 4 is **compulsory**.
Calculators may be used.
Read carefully the instructions on the answer book and make sure that the particulars required are entered on each answer book.

---

1. *This question is about the fundamental knowledge.*

(a) Give an example of instruction-level parallelism and discuss the factors that limit the further improvement of the instruction-level parallelism. [7]

(b) Describe what dependencies exist in the following sequence of statements. Explain how to remove the anti-dependency and the output dependency in these statements.

$a=b+c;\ d=a+c;\ d=a+e;\ b=2\times e;$ [10]

(c) Explain why superthreading and hyperthreading can improve the performance of the pipeline mechanism. [8]

(d) Top500 and Graph500 are two supercomputer lists in the world. Discuss the differences between these two lists. The discussions should focus on the type of performance that the two lists target, benchmarking applications and their features, and the key architecture factors in the computer systems that have impact on the performance. [10]

Continued

2. *This question is about parallel programming models*

(a) Discuss how synchronized statements can provide finer-grained synchronization than synchronized methods in Java. [8]

(b) Explain how to express a data to be sent in MPI communication operations, and discuss why *message tag* and *communicator* are necessary parameters in MPI_Send. [7]

(c) A collective communication operation is performed by all relevant processes at the same time with the same set of parameters. However, different processes may interpret the parameters differently. Describe, using illustrative examples if necessary, the operations of the following two MPI collective communication calls. Further, discuss how different processes interpret different parameters in these functions.

i) MPI_Bcast (*void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm*) [6]

ii) MPI_Scatter(void **sendbuf*, int *sendcnt*, MPI_Datatype *sendtype*, void **recvbuf*, int *recvcount*, MPI_Datatype *recvtype*, int *root*, MPI_Comm *comm*) [6]

(d) In MPI, the programmers are allowed to construct their own data types. MPI_Type_vector is one of the MPI functions serving this purpose. The format of the function is as follows:

```
MPI_Type_vector ( int count,
                  int blocklength,
                  int stride,
                  MPI_Datatype oldtype,
                  MPI_Datatype *newtype)
```

Let *oldtype* ={(MPI_DOUBLE, 0), (MPI_CHAR, 64)} with the extent of 72 bits.

Give the memory layout of *newtype* after calling

MPI_Type_vector (3, *2*, *4*, *oldtype*, *newtype*) [8]

Continued

3. *This question is about high performance computing systems.*

(a) Describe the differences between Clusters and Massively Parallel Processing (MPP) systems. [8]

(b) Discuss the differences in the designing objectives between a GPU and a CPU. [10]

(c) Describe what the data sieving technique and the collective I/O are. Discuss how the data sieving technique and collective I/O can help improve the I/O performance. [10]

(d) The topology of node interconnection plays an important role in the performance of a HPC system. Explain what *node degree* and *bisection width* are in a network topology. What are the node degree and bisection width of a binary tree? Draw an exemplar binary tree with the depth of the tree being 3. [7]

Continued

4. *This question is about performance analysis methods.* (**Note this question is compulsory**)

(a) Describe what Amdahl's law is. Discuss how (give an example if it helps the explanation) we can apply Amdahl's law to help parallelize a serial program. Further, discuss the limitation of using Amdahl's law to analyse the performance of a parallel program. [10]

(b) The program in the following listing is a parallel implementation that approximates π. Each parallel worker throws 'darts' at a unit circle within a unit square and then approximates the value of π by dividing the number of darts that fall within the circle by the total darts thrown and multiplying by four.

Devise a performance model to capture the execution time of this application. Clearly describe all parameters you use and measurements you would need to take. Further, discuss the scaling behaviour of the execution time over the parameters you use to construct the model. [20]

```
1  /* ----------------------------------------------------------
2   * Calculate pi using a "dartboard".
3   * ORIGINAL AUTHOR: Roslyn Leibensperger
4   * ---------------------------------------------------------- */
5
6  #include <stdlib.h>
7  #include <stdio.h>
8  #include "mpi.h"
9  double dboard (int darts);
10
11 #define DARTS 5000 /* number of throws at dartboard */
12 #define ROUNDS 10 /* number of times "darts" is iterated */
13 #define MASTER 0 /* task ID of master task */
14 #define sqr(x) ((x)*(x))
15
16 MPI_Status status;
17 MPI_Request request;
18
19 main(int argc, char **argv)
20 {
```


Continued

```
21      double homepi, /* value of pi calculated by current task */
22             pi, /* average of pi after "darts" is thrown */
23             avepi, /* average pi value for all iterations */

24      pirecv, /* pi received from worker */
25      pisum; /* sum of workers pi values */
26      int mytid, /* task ID - also used as seed number */
27      nproc, /* number of tasks */
28      source, /* source of incoming message */
29      mtype, /* message type */
30      msgid, /* message identifier */
31      nbytes, /* size of message */
32      rcode, /* return code */
33      i, n;
34
35      /* Obtain number of tasks and task ID */
36
37      MPI_Init(&argc, &argv);
38      MPI_Comm_rank(MPI_COMM_WORLD, &mytid);
39      MPI_Comm_size(MPI_COMM_WORLD, &nproc);
40      printf ("MPI task ID = %d\n", mytid);
41
42      /* Set seed for random number generator equal to task ID */
43      srandom (mytid);
44
45      avepi = 0;
46      for (i = 0; i < ROUNDS; i++) {
47
48          /* All tasks calculate pi using dartboard algorithm */
49          homepi = dboard(DARTS);
50
51          /* Workers send homepi to master */
```


Continued

```
52          /* - Message type will be set to the iteration count */
53          /* - A non-blocking send is followed by mpi_wait */
54          /* this is safe programming practice */
55
56          if (mytid != MASTER) {
57                  mtype = i;
58                  MPI_Isend(&homepi, 1, MPI_DOUBLE, MASTER, mtype,
59                                  MPI_COMM_WORLD, &request);
60                  MPI_Wait(&request, &status);
61          }
62
63          /* Master receives messages from all workers */
64          /* - Message type will be set to the iteration count */
65          /* a message can be received from any task, as long as the */
66          /* message types match */
67          /* - The return code will be checked, and a message displayed */
68          /* if a problem occurred */
69
70          else {
71                  mtype = i;
72                  pisum = 0;
73                  for (n = 1; n < nproc; n++) {
74                          MPI_Recv(&pirecv, 1, MPI_DOUBLE, MPI_ANY_SOURCE, mtype,
75                                          MPI_COMM_WORLD, &status);
76
77                          /* keep running total of pi */
78                          pisum = pisum + pirecv;
79                  }
80
81          /* Master calculates the average value of pi for this iteration */
82          pi = (pisum + homepi)/nproc;
83
```


Continued

```
84                /* Master calculates the average value of pi over all iterations */
85                avepi = ((avepi * i) + pi)/(i + 1);
86                printf(" After %3d throws, average value of pi = %10.8f\n",
87                      (DARTS * (i + 1)),avepi);
88                }
89          }
90          MPI_Finalize();
91  }
92
93  double dboard(int darts)
94  {
95          double x_coord, /* x coordinate, between -1 and 1 */
96                y_coord, /* y coordinate, between -1 and 1 */
97                pi, /* pi */
98                r; /* random number between 0 and 1 */
99                int score, /* number of darts that hit circle */
100               n;
101         unsigned long cconst; /* used to convert integer random number */
102         /* between 0 and 2^31 to double random number */
103         /* between 0 and 1 */
104
105         cconst = 2 << (31 - 1);
106         score = 0;
107
108         /* "throw darts at board" */
109
110         for (n = 1; n <= darts; n++) {
112               /* generate random numbers for x and y coordinates */
113               r = (double)random()/cconst;
114               x_coord = (2.0 * r) - 1.0;
115               r = (double)random()/cconst;
116               y_coord = (2.0 * r) - 1.0;
```


Continued

```
117
118             /* if dart lands in circle, increment score */
119             if ((sqr(x_coord) + sqr(y_coord)) <= 1.0)
120                     score++;
121     }
122
123     /* calculate pi */
124     pi = 4.0 * (double)score/(double)darts;
125     return(pi);
126 }
```


End