ITCS 4/5145 Parallel Computing
Test 1 5:00 pm - 6:15 pm, Wednesday February 17, 2016

Solutions

Name: ....

Answer questions in space provided below questions. Use additional paper if necessary but make sure your name is on additional sheets. *Clearly show how you obtained your answers. (No points for simply writing a numeric answer without showing how you got the answer, even if correct.)* This is a closed book test. Do not refer to any materials except those supplied for the test. Supplied: *"Summary of OpenMP 3.0 C/C++ Syntax."* *and "Basic MPI routines."*

Total /40
7 pages

Qu. 1 Answer each of the following briefly:

(a) According to Amdahl's law, what the maximum speed-up of a parallel computation given that 90% of the computation can be divided into parallel parts? What assumption are you making? 2

Max speedup = 1/f = 1/0.10 = 10 with infinite number of processes.

(b) How can one make five threads do exactly the same code sequence using OpenMP? 2

```
#pragma omp parallel num_threads(5)
{
… // code
}
```

(c) In the following OpenMP code sequence, which variable or variables must be declared as private variables (in a private clause):

```
int x, tid, a[100];
#pragma omp parallel
{
    tid = omp_get_thread_num();
    n = omp_get_num_threads();
    a[tid] = 10*n;
}
```

2

tid and n.

(d) What does the nowait clause do in the Open sections directive? 2

Threads do not wait after finishing their section

(e) What is the value of sum after the following OpenMP code sequence, i.e. what number does the printf statement print out? 2

```
int i, sum;
omp_set_num_threads(2);
sum = 0;

#pragma omp parallel for reduction(+:sum)
for (i = 0; i < 5; i++ ) {
    sum++;
}
printf("Sum = %d\n", sum);
```

5

(f) If x is a shared variable initialized to zero and three concurrent threads execute the statement x = x + 1; what are the possible value of x afterwards? 2

All threads separate without any overlap. x = 1 + 1 + 1 = 3

All threads read x before any have altered it and last writes back x = 1

Two threads read x after one writes back a 1, and these threads write 2

So 1, 2 or 3

(g) Identify all the dependencies in the following sequence:

1. **a = b + c;**
2. **y = a;**
3. **a = 3;**
4. **x = y + z;**

Clearly show how you got your answer. The statements are numbered so that you can refer to them. 2

1 and 2 read after write (true) dependency
1 and 3 write after write (output) dependency
2 and 4 read after write (true) dependency
2 and 3 write after read dependency (antidependency)

(h) In the MPI statement:

MPI_Send(&x,1,MPI_INT,1,msgtag, MPI_COMM_WORLD);

what can be inferred about how x has been declared. 2

It is a single integer, declared as int x;

(i) In MPI, how does the programmer specify how many processes the program will use? 2

When you execute the program with mpiexec using the -n option to specify the number processes, e.g.. mpiexec –n 4 prog1

(j) When does the MPI routine MPI_Send() return? 2

When its local actions have completed and it is safe to alter the arguments but the message may not have been received.

(k) What is a Jacobi iteration? 2

Values are computed using only values from the previous iteration.

(l) What does the -l option specify when used with the Linux C compiler (gcc or cc)? Give an example. (The -l option was used in Assignment 2, but the question is asking what -l specifies in general.)

Specifies a library. –lm, -lX11

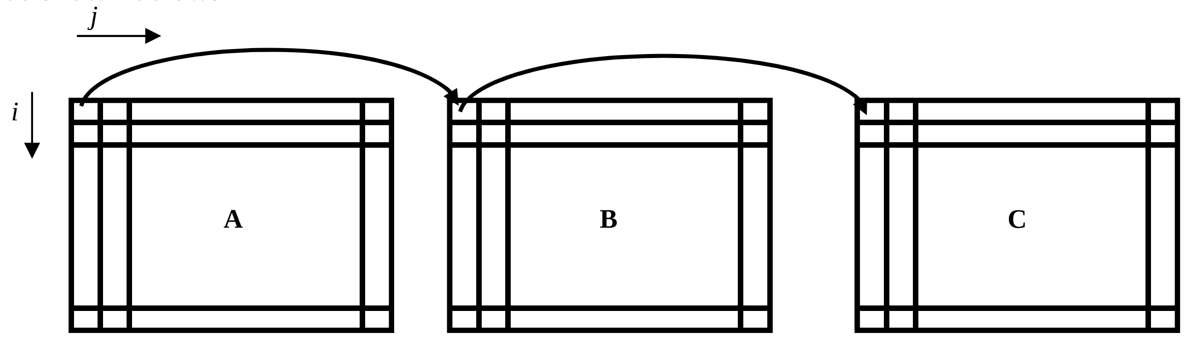
2

The following include statements can be assumed:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <omp.h>
#include "mpi.h"
```

in Qu 2 a, b and c. However any define statements, variables, and arrays that you use must be declared.

Qu. 2 (a) Write a sequential C program to perform matrix addition adding two matrices **A[N][N]** and **B[N][N]** to produce a matrix **C[N][N]**. The arrays hold doubles. **N** is a constant defined with a define statement and set to 256. In matrix addition, the corresponding elements of each matrix are added together to form elements of result matrix, as shown below:



i.e. given elements of **A** as $a_{i,j}$ and elements of **B** as $b_{i,j}$, each element of **C** computed as $c_{i,j} = a_{i,j} + b_{i,j}$. You can assume that the arrays are initialized with values but show where that would be in the program with comments.

4

```
#define N 256
int main(int argc, char *argv[]) {
    int i, j;
    double A[N][N], B[N][N], C[N][N];
    …                              // initialize arrays

    for (i = 0; i < N; i++) {
        for (j = 0; j < N; j++)   {
                C[i][j] = A[i][j] + B[i][j];
        }
    }
    …
    return 0;
}
```

(b) Modify the program in 2(a) become an OpenMP program performing matrix addition using ***T*** threads where ***T*** is a constant defined with a define statement and set to 16.

4

```
#define N 256
#define T 16
int main(int argc, char *argv[]) {
    int i, j;
    double A[N][N], B[N][N], C[N][N];
    … initialize arrays

    omp_set_num_threads(T);        //set number of threads here

    #pragma omp parallel for private(j)
    for (i = 0; i < N; i++) {                // parallel matrix addition
            for (j = 0; j < N; j++)   {
                C[i][j] = A[i][j] + B[i][j];
            }
    }
    return 0;
}
```

(c) Modify the program in 2(a) to become an MPI program performing matrix addition using ***P*** processes where ***P*** is determined when the code is executed. Partition the matrices into **blksz** rows, where **blksz** = ***N***/***P*** (Clue: remember how matrix multiplication was done.) It can be assumed that ***N***/***P*** is an integer.

8

```
#define N 256
int main(int argc, char *argv[]) {
    int i, j;
    double A[N][N], B[N][N], C[N][N];

    MPI_Status status;        // MPI variables
    int P, blksz;

    MPI_Init(&argc, &argv);    // Start MPI
    MPI_Comm_size(MPI_COMM_WORLD, &P);
    blksz = N/P;

    // Scatter input matrix A
    MPI_Scatter(A, blksz*N, MPI_DOUBLE, A, blksz*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    // Scatter input matrix B
    MPI_Scatter(B, blksz*N, MPI_DOUBLE, B, blksz*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    for(i = 0 ; i < blksz; i++) {
        for(j = 0 ; j < N ; j++) {
            C[i][j] = A[i][j] + B[i][j];
        }
    }

    // gather results
    MPI_Gather(C, blksz*N, MPI_DOUBLE, C, blksz*N, MPI_DOUBLE, 0, MPI_COMM_WORLD);

    MPI_Finalize();
    return 0;
}
```