Parallel Computing
Test 1
5:00 pm - 6:15 pm, Thursday Feb 20, 2014

With solutions (briefly)

Name: ....

This is a closed book test. Do not refer to any materials except those supplied for the test.

Supplied: "*Basic MPI routines*" and "*Summary of OpenMP 3.0 C/C++ Syntax.*"

Answer questions in space provided below questions. Use additional paper if necessary but make sure your name is on additional sheets.

Total /40

Qu. 1 Answer each of the following briefly:

(a) What is the maximum speed-up of a parallel computation given that 90% of the computation can be divided into six equal parts that can be executed at the same time?

2

Sequential = 100
Parallel = 10 + 90/6 = 10 + 15 = 25
Speedup = 100/25 = 4

Can get the same answer by using Amdahls' law:

$$S(p) = \frac{p}{1 + (p-1)f} = \frac{6}{1 + 5\ \times 0.1} = 4$$

Note this question does not ask the maximum speed up with an infinite number of processors (S(∞) = 10)

(b) Give one advantage for using parallel patterns. 2

From lecture notes:

- Abstracts/hides underlying computing environment
- Generally avoids deadlocks and race conditions
- Reduces source code size (lines of code)
- Leads to automated conversion into parallel programs without need to write with low level message-passing routines such as MPI.
- Hierarchical designs with patterns embedded into patterns, and pattern operators to combine patterns.
-

(c) Give one disadvantage for using patterns. 2

From lecture notes:

- New approach to learn
- Takes away some of the freedom from programmer
- Performance reduced (c.f. using high level languages instead of assembly language)

(d) In the Seeds framework a parameter called “segment” appears in the DiffuseData and GatherData methods. How is this parameter used for matrix addition? 2

Strictly, identifies the particular slave in a workpool.

(e) There are three versions of the Java based Seeds Framework. Which did you use for Assignment 1? 2

Multicore version

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

When its local actions have been completed and the message is safely on its way (local variables can be altered) but before the message has been received.

(g) In Assignment 2, which command did you use to compare the output of a sequential C program with that of a MPI program? 2

diff command

(h) In Assignment 2, how did you specify the computers to use on the UNC-C cluster? 2

Machines file specified with the –machines option.

(i) What is the fundamental difference between a process and a thread? 2

Threads share memory and some resources.

(j) What is a detached thread? Suggest how one might create a detached thread in OpenMP.
(Not said specifically in lecture notes) 2

A join is not used in the parent thread to wait for thread to complete. When the thread completes, its resources are deleted. (1 point)

In OpenMP, using the no-wait clause

(k) What is a critical section? 2

A section of code that one thread/process can enter at a time. Used to control access to shared resources

(l) In OpenMP, how can one make five threads do completely different code sequences? 2

Use the sections directive

Supplied: "*Basic MPI routines*" and "*Summary of OpenMP 3.0 C/C++ Syntax.*"

In the following, provide comments in your code to help the grader. ***If I do not understand the code, I will assume it is incorrect.*** The programs should be *complete programs* of the form:

```
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#include <omp.h>
main(int argc, char **argv ) {
   …
return(0)
}
```

You are to have all required statements for a working program.

Qu. 2 (a) First write a sequential C program to compare two integer *N* x *N* arrays and report how many elements are different (i.e. if A[i][j] is different to B[i][j], for all *i* and *j*) with a print statement. *N* is a defined constant set to 1000. You can assume the arrays are initialized with values but show where that would be in the program with comments.

4

```
#define N 1000
#define P ?
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#include <omp.h>
main(int argc, char **argv ) {

int A[N]N], B[N][N];
int count = 0;
                // initialize arrays

for (i = 0, i < N; i++)
 for (j = 0; j < N; j++)
   if (A[i][j] != B[i][j]) count++;

printf("The number of different elements is %d\n",count);

return(0);
}
```

(b) Modify the code using OpenMP directives to parallelize the code using *P* threads, where *P* is also a defined constant and *N* is a multiple of *P*.

6

```
#define N 1000
#define P ?
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#include <omp.h>
main(int argc, char **argv ) {

int A[N]N], B[N][N];
int count = 0;
                     // initialize arrays
                    // set number of threads to P
omp_set_num_threads(P); // could be done with num_threads clause in parallel directive

#pragma omp parallel for reduction(+:count)
  for (i = 0, i < N; i++)
    for (j = 0; j < N; j++)
     if (A[i][j] != B[i][j]) count++;

printf("The number of different elements is %d\n",count);

return(0);
}
```

There are other possible solutions.

(c) Finally re-write the code to be an MPI program with *P* processes.

6

```
#define N 1000
#define P ?
#include <stdio.h>
#include <stdlib.h>
#include "mpi.h"
#include <omp.h>
main(int argc, char **argv ) {   // Note: All processes execute same code

int A[N]N], B[N][N];
int rank;               // my rank
int count = 0, result;
int blk = N/P;         // block size    Note: N a multiple of P

                // initialize arrays

MPI_Bcast(A, N*N, MPI_INT, 0, MPI_COMM_WORLD); // broadcast A
MPI_Bcast(B, N*N, MPI_INT, 0, MPI_COMM_WORLD); // broadcast B

MPI_Comm_rank(MPI_COMM_WORLD,&rank); // find rank

start = rank*blk; end = start + blk;

  for (i = start, i < end; i++)
    for (j = 0; j < N; j++)
     if A[i][j] == B[i][j] count++

MPI_Reduce (count, result, N, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

printf("The number of different elements is %d\n",result);

return(0);
}
```

There are other solutions, including using scatter just that part each array required to each process. Then do not need start and end.