ITCS 4145/5145 Parallel Programming
Final exam – 2 hours

UNC-C students: 11 am - 1:00 pm, Tuesday December 10th 2013

Name: .....

12 pages

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

Provided: Information on some MPI routines.

Answer each question in space provided below the question. Use additional paper if necessary but make sure your name is on every additional sheet.

Total /60

Do not refer to any materials for this part

Qu. 1 Answer each of the following briefly: (40)

a) Suppose you have a quad-core computer that can execute four processes simultaneously (one process on each core). Suppose there is a program in which 40% of total execution time must be executed sequentially but the remainder can be divided into 6 equal parts that can be executed concurrently. What is the maximum speedup you can get using all four cores over using one core. Clearly explain your answer. No points for simply putting down a numerical answer without an explanation, even if correct.

2

b) What is meant by pattern programming?

2

c) In the Seeds framework, what does the bootstrap class do?

2

d) In Assignment 1 (Seeds framework), you are asked to provide the name of your computer. How are you instructed to find out that name?

2

e) Why is there no concept of shared and private data when using the Paraguin compiler?

2

f) In Assignment 2 (Paraguin assignment) why did the second matrix need to be broadcast instead of scattered?

2

g) Parallelize the following code using the scatter/gather pattern in the Paraguin compiler:

```
double a[N], b[N];
int i;

// b is initialized with data from a file

for (i = 0; i< N; i++) {
        a[i] = 3*b[i] – 5*i + 13;
}
```

2

h) When does the MPI routine MPI_SSend() return?

2

i) In Assignment 3 (MPI, Part 3 matrix multiplication) you are asked to read a file containing two 512x512 matrices of floating-point numbers as input. The assignment did not say how to specify the file name. How did you specify the file name for your program?

2

j) As a programmer, if you use a routine known to be *not* thread-safe, what should you avoid? 2

k) Rewrite the following loop nest so that all iterations of BOTH loops will be run in parallel among multiple processors in OpenMP. 2

```
for (i = 0; i< N; i++) {
        for (j = 0; j < M; j++) {
                …
        }
}
```

l) Using Bernstein conditions for parallelism, determine whether 2

```
for (int i = 0; i<5; i++) A[i] = A[i*2];
```

can be written as:

```
forall (int i = 0; i<5; i++) A[i] = A[i*2];
```

m) How are threads created in Java (one way)? Give code. 2

n) How would you compile a program that included both OpenMP and CUDA code? 2

o) How does the Barnes-Hut algorithm reduce the time complexity of the *N*-body problem? 2

p) Suppose a pipeline is constructed to compute more than one instance of a problem. What is the speed up with 100 instances of the problem and 25 pipe stages compared to executing one instance?

2

q) What is the parallel time complexity of Odd-Even Transposition Sort with N numbers and N processors? (Proof not necessary.)

2

r) Suppose it is necessary to create barrier that causes all threads in a CUDA kernel to wait until they reach that point before continuing. How can that be achieved in a CUDA program having N thread blocks where N is greater than 1?

2

Qu. 2 (a) Write a complete MPI program to find how many zeros there are in an integer array A[N] using a scatter/gather pattern. Define N as a constant with a value 10,000. Define the number of slaves as P and set to 10 slaves. Each slave is sent one group of N/P numbers from the array.

7

Provide very clear explanation of how the program works, and comments in your code. If I do not understand the code, I will assume it is incorrect.

You may refer to the information on MPI routines at the end of the test.

2(b) Repeat 2(a) but using OpenMP sharing the work across 10 threads?

7

Qu 3 Write a *complete* CUDA program that will sort numbers in an integer array A[N] using the Rank sort algorithm, such that each CUDA thread handles one number and places it in the correct location in a sorted list B[N]. Define N as a constant and set to 1000.Use the following code to load A[N] with numbers:

```
for (i = 0; i< N; i++)
  A[i] = (int) rand();
```

You can assume that all numbers are different. Also sort the number using ranksort on the host CPU and measure the execution time in each case. Compute the speedup factor.

The thread ID can be computed from:

```
int tid = threadIdx.x + blockDim.x * blockIdx.x;
```

Provide very clear explanation of how the program works, and comments in your code. If I do not understand the code, I will assume it is incorrect.

10

Space to continue your answer for Qu 3:

Some MPI routines

int MPI_Bcast(void *buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)

Input/Output Parameter
buffer -- starting address of buffer

Input Parameters
count -- number of entries in buffer
datatype -- data type of buffer
root -- rank of broadcast root
comm -- communicator

int MPI_Comm_rank(MPI_Comm comm, int *rank)

Input Argument
comm -- communicator

Output Argument
rank -- rank of the calling process in the group of comm

int MPI_Comm_size(MPI_Comm comm, int *size)

Input Parameter
comm -- communicator

Output Parameter
size -- number of processes in the group of comm

int MPI_Finalize(void)

int MPI_Gather(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)

Input Parameters
sendbuf -- starting address of send buffer
sendcount -- number of elements in send buffer
sendtype -- data type of send buffer elements
recvcount -- number of elements for any single receive
recvtype -- data type of recv buffer elements
root -- rank of receiving process
comm -- communicator

Output Parameter
recvbuf -- address of receive buffer

int MPI_Init(int *argc, char ***argv)

Input Parameters
argc -- Pointer to the number of arguments
argv -- Pointer to the argument vector

int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm comm, MPI_Status *status)

Output Parameters

buf -- initial address of receive buffer
status -- status object

Input Parameters

count -- maximum number of elements in receive buffer
datatype -- datatype of each receive buffer element
source -- rank of source
tag -- message tag
comm -- communicator

int MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype datatype, MPI_Op op, int root, MPI_Comm comm)

Input Parameters

sendbuf - address of send buffer
count - number of elements in send buffer
datatype - data type of elements of send buffer
op - reduce operation
root - rank of root process
comm - communicator

Output Parameter

recvbuf - address of receive buffer

int MPI_Scatter(void *sendbuf, int sendcnt, MPI_Datatype sendtype, void *recvbuf, int recvcnt, MPI_Datatype recvtype, int root, MPI_Comm comm)

Input Parameters

sendbuf - address of send buffer
sendcount - number of elements sent to each process
sendtype - data type of send buffer elements
recvcount - number of elements in receive buffer
recvtype - data type of receive buffer elements
root - rank of sending process
comm - communicator

Output Parameter

recvbuf - address of receive buffer

int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)

Input Parameters

buf -- initial address of send buffer
count -- number of elements in send buffer
datatype -- datatype of each send buffer element
dest -- rank of destination
tag -- message tag
comm -- communicator