ITCS 4145/5145 Parallel Programming
Final exam – 2 hours
With solutions

UNC-C students: 11 am - 1:00 pm, Tuesday December 10th 2013
UNC-W students: 11 am - 1:00 pm, Tuesday December 10th 2013
UNC A&T students: 1 pm - 3 pm, Thursday December 12, 2013
ECU students: 11 am - 1:00 pm, Tuesday December 5th 2013
UNC-G students: 12 noon- 2:00 pm, Tuesday December 10th 2013

Name: ....

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

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

Sequential = 0.4 + 0.6 = 1.0 (.4 + .1 + .1 + .1 + .1 + .1 + .1 = 1.0)
Parallel = 0.4 + 2 x 0.1 = 0.6 (takes 0.1 to do first 4 parts and another 0.1 to do the remaining 2 parts)
(.4 + .1 (4 parts concurrently) + .1 (2 parts concurrently) = .6)
Speed up = 1.0/0.6 = 1.667

b) What is meant by pattern programming?

2

Programmer begins by constructing his program using established computational or algorithmic "patterns" that provide a structure. Higher level tools can then be sued to create executable code avoiding low level tools such as MPI or OpenP

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

2

Starts the framework and runs the user (module) program.

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

Run the hostname command on the (DOS) command line.

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

2

Because it creates a parallel program that runs on a distributed-memory system. No data is shared. All data is private.

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

2

Because every processor needed the entire matrix (all rows and columns).

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

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

// b is initialized with data from a file

#pragma paraguin begin_parallel
#pragma paraguin bcast b
#pragma paraguin forall
for (i = 0; i< N; i++) {
        a[i] = 3*b[i] – 5*i + 13;
}
#pragma paraguin gather a
#pragma paraguin end_parallel
```

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

2

When the message has been received.

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

In the job description file at the end of the mpirun line. (Could also be specified at the end of the qsub command.)

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

2

Call the routine from multiple threads that might occur at the same time.
(Thread safe routines can be called from multiple threads at the same time and always produce correct results.)

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++) {
                …
        }
}
```

```
#pragma omp parallel for private(p, i, j)
for (p = 0; p < N * M; p++) {
        i = p / N;
        j = p % N;
        …
}
```

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];
```

Unfold loop:

```
        A[0] = A[0];
        A[1] = A[2];
        A[2] = A[4];
        A[3] = A[6];
        A[4] = A[8];
```

No. Read/write dependency A[4]

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

2

```
public class HelloThread extends Thread {
	public void run() {
		System.out.println("Hello from a thread!");
	}
	public static void main(String args[ ]) {
		HelloThread myThread = new HelloThread();
		myThread.start();
	}
}
```

Or

```
public class HelloRunnable implements Runnable {
	public void run() {
		System.out.println("Hello from a thread!");
	}
	public static void main(String args[ ]) {
		HelloRunnable myThread = new HelloRunnable();    // Runnable object
		Thread tr = new Thread(myThread);                 // Create Thread object
		tr.start(); // Start thread and execute run method
	}
}
```

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

2

```
nvcc –fopenmp –o <exec_name><source_file> -I/usr/local/cuda/include –L/usr/local/cuda/lib –lcuda –lcudart
```

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

2

Approximates a cluster of distant bodies as a single distant body with mass sited at the center of mass of the cluster. Space divide into sub-cubes with one in each. Create an octtree. Leaves represent cells each containing one body. Total mass and center of mass of subcube stored at each vertex (node).Force on each body obtained by traversing tree starting at root, stopping at a node when the clustering approximation can be used, e.g. when r is greater than some distance D.

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 executed one instance.

2

m = 100
p = 25

speedup s(m) = ts/tp = (p * m) / (p+m-1) = 2500/124 = 20.16

q) What is the parallel time complexity of Odd-Even Transposition Sort with N numbers and N processors

2

O(N)

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

Cannot use __syncthreads() as this only synchronizes threads in a block.
Need to return to the main host program and insert a cudaThreadSynchronize();

Qu. 2 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.

MPI

```
#include <mpi.h>
#define N 10000
int main (int argc, char *argv[])
{
        int A[N], B[N];                         // actually B need only have N/P locations. Could malloc
        int i, rank, answer, P = 10, totalZero = 0;
        int chunksize = (int) ceil( ((double)N) / P);
        MPI_Status status;

        MPI_Init (&argc, &argv);
        MPI_Comm_rank (MPI_COMM_WORLD, &rank);
        MPI_Comm_size( MPI_COMM_WORLD, &P);

        MPI_Scatter(A, chunksize, MPI_INT, B, chunksize, 0, MPI_COMM_WORLD); // includes master here

        totalZero = 0;
        for (i = 0; i < chunksize && rank * chunksize + i < N; i++) {
                totalZero += (B[i] == 0);
        }

        MPI_Reduce(&totalZero, &answer, 1, MPI_INT, MPI_SUM, 0, MPI_COMM_WORLD);

        // Now we have the answer
        printf (“Number of zeros is %d\n”, answer);
        MPI_Finalize()
}
```

2(b) Repeat 2(a) but using OpenMP sharing the work across 10 threads? 7

OpenMP

```
#define N 10000
#define P 10

int main (int argc, char *argv[])
{
        int A[N], d;
        int i, tid, totalZero = 0;

        // A is initialized with values

        #pragma omp parallel num_threads(10)
        {

                #pragma omp for (dynamic, 1) reduction (+: totalZero)
                for (i = 0; i< N; i++) {
                        totalZero += (A[i] == 0);
                }

        }

        // Now we have the answer
        printf ("Number of zeros is %d\n", totalZero);
}
```

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

```
#include <stdio.h>
#include <cuda.h>
#include <stdlib.h>
#define N 1000                         // set N to some value

__global__ void gpu_ranksort(int *a, int *b, int N) {  // a is unsorted array, b is sorted array
    inti, x;
    inttid = threadIdx.x + blockDim.x * blockIdx.x;

    while(tid< N) {
        x = 0;
        for (i = 0; i< N; i++)        // count number less than it
            if (a[tid] > a[i]) x++;
        b[x] = a[tid];                // copy number into correct place
        tid += blockDim.x * gridDim.x;
    }
}

voidcpu_ranksort(int *a, int *b, int N) {  // a is the unsorted array, b is the sorted array
    int i, x, t;

    for (t = 0; t < N; t++) {         // for each number
        x = 0;
        for (i = 0; i< N; i++)        // count number less than it
            if (a[t] > a[i]) x++;
            b[x] = a[t];              // copy number into correct place
        }
}

int main(int argc, char *argv[])  {

    inti;                             // loop counter

    int Block_Dim_x;                  //Block structure values
    int Grid_Dim_x;                   // no of blocks in grid
    int noThreads_x;                  // number of threads available in device, one dimension

    int a[N],b[N],c[N];               // a the input array, b the sorted array on GPU, c the sorted array on
host
    int *dev_a, *dev_b;
    int size;                         // number of bytes in arrays

    cudaEvent_t start, stop;          // using cuda events to measure time
    float cpu_elapsed_time_ms, gpu_elapsed_time_ms;  // which is applicable for asynchronous code also
    for(i=0;i<N;i++)              // load array with some numbers
        a[i] = (int)rand();

/* ------------- SORTING DONE ON GPU ----------------------------*/

    cudaMalloc((void**)&dev_a, size);     // allocate memory on device
    cudaMalloc((void**)&dev_b, size);

    cudaMemcpy(dev_a, a , size ,cudaMemcpyHostToDevice);
    cudaMemcpy(dev_b, b , size ,cudaMemcpyHostToDevice);

    cudaEventCreate(&start);              // instrument code to measure start time
    cudaEventCreate(&stop);

    cudaEventRecord(start, 0);
//  cudaEventSynchronize(start);      // Needed?

    gpu_ranksort<<<Grid_Dim_x, Block_Dim_x>>>(dev_a,dev_b,N);

    cudaMemcpy(b,dev_b, size ,cudaMemcpyDeviceToHost);

    cudaEventRecord(stop, 0);         // instrument code to measure end time
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&gpu_elapsed_time_ms, start, stop );

    printf("Time for ranksort on GPU: %f ms.\n", gpu_elapsed_time_ms);  // print out execution time

/* ------------- SORTING DONE ON HOST CPU USING RANKSORT----------------------------*/

    cudaEventRecord(start, 0);        // use same timing
//  cudaEventSynchronize(start);      // Needed?

    cpu_ranksort(a,c,N);              // sort the list using ranksort

    cudaEventRecord(stop, 0);         // instrument code to measue end time
    cudaEventSynchronize(stop);
    cudaEventElapsedTime(&cpu_elapsed_time_ms, start, stop );

    printf("Time for ranksort on CPU: %f ms.\n", cpu_elapsed_time_ms);  // print out execution time

/* ------------------- check device creates correct results -----------------*/

    for(i=0;i <N;i++) {
        if (c[i] != b[i]) printf("ERROR in results, CPU (ranksort) and GPU (ranksort) create different
answers\n");
        break;
    }

/* --------------------- SPEED-UP FACTOR  --------------------------------*/

    printf("Speed up factor = %f\n",cpu_elapsed_time_ms/gpu_elapsed_time_ms);

/* --------------  clean up  ---------------------------------------*/

    cudaFree(dev_a);
    cudaFree(dev_b);

    cudaEventDestroy(start);
    cudaEventDestroy(stop);

    return 0;
}
```

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