ITCS 4/5145 Parallel Computing
Test 2
5:00 pm - 6:15 pm, Thursday April 10, 2014

Name: ....

This is a closed book test. Do not refer to any materials except those supplied for the test. 6 pages incl. one blank page

Appendix attached *"Brief Summary of CUDA"* (Page 6)

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 in the spaces provided:

(a) Why are statements of a program are not necessarily executed in the order as specified by the programmer? 2

(b) Use Bernstein's conditions to determine whether the following sequence can be executed in parallel, totally or partially:

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

Clearly show how you got your answer. (No marks for just yes or no!) 4

(c) Why does false sharing reduce performance of shared memory programs?
(This question is not asking what is false sharing. No points for that.) 4

(d) What meant by hybrid parallel programming? 2

(e) In the Barnes Hutt algorithm, a tree is created. What is stored at each vertex of this tree? 2

(f) What is meant by the term "Jacobi" iteration as opposed to other types of iteration that are not Jacobi?

2

(g) Explain the red-black ordering (iteration) method. Is it necessary to store both the current iteration values and next iteration values? Explain. 4

(h) What is the sequential time complexity of Odd-Even Transposition Sort with *N* numbers? What is the parallel time complexity with *N* processors?

4

(i) Demonstrate how to sort the sequence: 4

7 2 8 5

using Batchers' Bitonic Mergesort algorithm. Clearly show the position of each number after each step.

Qu. 2 Write a CUDA program that implements the iterative sequential code below:

```
#define N 1000
...
main {

   int i, iteration, limit = 50000;
   int g[N], h[N];

   // assume array h initialized in host
   ...

   for (iteration = 0; iteration <  limit; iteration++) {
      for (i = 1; i < N-1; i++)
         g[i] = 0.5*(h[i-1]+h[i+1]);
   }

   for (i = 1; i < N-1; i++) { // print out g[1] to g[N-2] on host
      printf{%d \n”, g[i]);
   }
...
}
```

Use a 1-dimensional grid and block structure. Set the number of CUDA blocks in a grid to 1 and number of threads to 1200. Some threads will not be used but you need to handle that in the code.

Provide comments in your code to help the grader! Briefly describe your method. ***If I do not understand the code, I will assume it is incorrect.*** Include statements need not be given in the answer.

See ***“Brief Summary of CUDA for Test”*** on last page for reference.

12

Intentionally blank to continue the answer for Qu 2.

"Brief Summary of CUDA"

Information that may or may not be needed

Function Qualifers

__global__	called from host, executed on device
__device__	called from device, executed on device

Variable Qualifiers (Device)

__device__	variable on device (Global Memory)

Built-in Variables (Device)

dim3 gridDim	dimensions of the current grid (gridDim.x, . . .)
dim3 blockDim	dimensions of the current block (composed of threads)
uint3 blockIdx	block location in the grid (blockIdx.x, . . .)
uint3 threadIdx	thread location in the block (threadIdx.x, . . .)

Thread ID

1-D int tid = threadIdx.x + blockDim.x * blockIdx.x;

Host / Device Memory

cudaMalloc(&devptr, size)	Allocate Device Memory
cudaFree(devptr)	Free Device Memory
cudaMemcpy(dst, src, size, cudaMemcpyKind kind)	Transfer Memory kind=cudaMemcpyHostToDevice, cudaMemcpyDeviceToHost,...

Synchronizing

syncthreads()	Synchronizing one Block (device call)
cudaDeviceSynchronize()	Synchronizing all Blocks (host call)

Kernel

kernel<<<dim3 blocks, dim3 threads[, ...]>>>(arguments)	Kernel launch