CS4961: Parallel Programming Midterm Quiz
December 1, 2009

Instructions:
This is a take-home open-book, open-note exam. The format is that of a long homework. The exam is due by 5PM, Tuesday, December 15. Use the CADE handin program, as follows:
The goal of the exam is to reinforce your understanding of issues we have studied in class.

CS4961: Parallel Programming
Midterm Quiz
December 1, 2009

I. Definitions (30 points)

Provide a very brief definition of the following terms:

a. Locality

b. Data reuse

c. Global view languages

d. Divergent branches

e. Broadcast

f. Global memory coalescing

g. One-sided communication

h. Performance counters

i. Memory bank conflicts

j. Domain

III. Problem Solving (60 points)

In this set of six questions, you will be asked to provide code solutions to solve particular problems. This portion of the exam may take too much time if you write out the solution in detail. I will accept responses that sketch the solution, without necessarily writing out the code or worrying about correct syntax. Just be sure you have conveyed the intent and issues you are addressing in your solution.

a. Chapter 7, #7 in textbook. Use MPI to implement Batcher's Bitonic Sort, described in Chapter 4, and use *blocking* send/receive. Assume you have as input the results of #4 to initialize the computation.

b. Chapter 7, #8 in textbook. Revise the Batcher Bitonic Sort of (b) to overlap communication and computation by using non-blocking sends/receives.

c. Given the following sequential code, sketch out a CUDA implementation. Derive a partitioning into threads and blocks that does not exceed various hardware limits. Assume all data is stored in global memory.

```
…
float a[1024][1024], b[1024];

for (i=0; i<1024; i++)
   for (j=0; j<1024-i; j++)
      b[i+j] += arbitrary_function(a[i][j]);
```

d. Given the following CUDA code, add synchronization to derive a correct implementation that has no race conditions. (Hint: You should be able to simply insert __synchthreads() calls without modifying the code.)

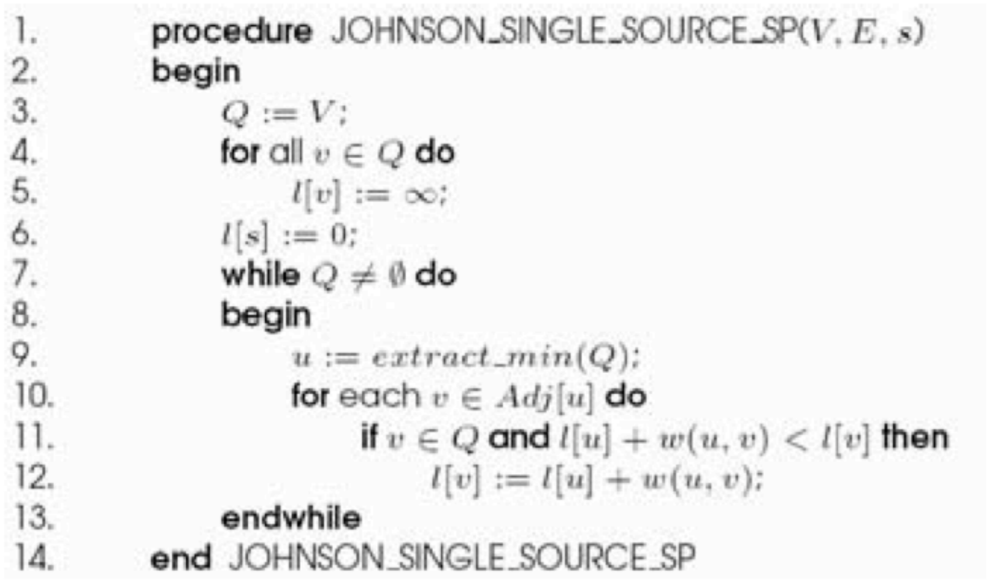
```
__global__ compute (float *a, float *b, int BLOCKSIZE) {
    __shared__ s_a[128], s_b[128];
    /* copy portion of input data into shared memory */
    s_a[threadIdx.x] = a[blockIdx.x*BLOCKSIZE + threadIdx.x];

    /* Time step loop */
     for (int t = 0; t<MAX_TIME; t++) {
        /* alternate inputs and outputs on even/odd time steps */
        if (t % 2 == 0) {
           int boundary = min((blockIdx.x+1)*BLOCKSIZE-1,
                                  blockDim.x*BLOCKSIZE-1,threadIdx+2);
           s_b[threadIdx.x] = s_a[threadIdx.x] + s_a[boundary];
        }
        else /* (t%2 == 1) */ {
           int boundary = min((blockIdx.x+1)*BLOCKSIZE-1,
                                  blockDim.x*BLOCKSIZE-1,threadIdx+2);
           s_a[threadIdx.x] = s_b[threadIdx.x] + s_b[boundary];
       }
  }

/* Result is in s_b, and must be copied to b */
b[blockIdx.x*BLOCKSIZE + threadIdx.x] = s_b[threadIdx.x];

}
```

e. Sketch out a parallel solution to the Johnson's single source shortest path algorithm using the CRS representation of sparse matrices from P2.



```
procedure JOHNSON_SINGLE_SOURCE_SP($V, E, s$)
begin
    $Q := V$;
    for all $v \in Q$ do
        $l[v] := \infty$;
    $l[s] := 0$;
    while $Q \neq \emptyset$ do
    begin
        $u := extract\_min(Q)$;
        for each $v \in Adj[u]$ do
            if $v \in Q$ and $l[u] + w(u, v) < l[v]$ then
                $l[v] := l[u] + w(u, v)$;
    endwhile
end JOHNSON_SINGLE_SOURCE_SP
```

f. Create a domain in Chapel that represents the even elements of a 2-dimensional array and iterate over it in parallel to find the product of these elements and constant 3.14159.

III. Essay Question (10 points)
Write a very brief essay describing the parallel programming process. Your answer can be a combination of a summary of the contents of Chapter 9, other material in this course, and your own personal experience. How does it differ from writing sequential code? Why is performance so important, and how is it achieved?