Name: _____

KUID: _____

Do not log into our regular class zoom session. We will not be meeting today.

This exam is written as if it were a normal fifty-minute exam. You will have from 9:00 am – 12:00 noon to finish it, simply as a precaution against any unforeseen problems. You are encouraged to finish it as promptly as possible so that you can be assured it can be submitted by noon. No extensions will be given.

Exam 2 Ground Rules

1. This exam is closed book, closed notes, and closed computer (other than a word processor to write your answers). You are not to communicate in any way with any person other than me while working on this exam. You cannot use your computer to compile or run OpenMPI, CUDA, or OpenCL code of any sort. Nor can you use it to log in to a remote machine for any purpose.
2. You cannot access google or any other remote resource accessible over the internet, including the class web site and links contained therein.
3. The only exception to these two rules is that you are allowed to email me questions you may have regarding the exam while taking it, and you are of course allowed to read my responses back to you. I will be most responsive from 9:00-11:00; I have another class at 11:00, so responses during the final hour that you have to work on the exam may or may not come to you in time.

   My email address is: jrmiller@ku.edu

I assert that I have not violated any of the Ground Rules listed above and that I have not had any communication of any sort with anyone else during the time period allotted for the exam. The work represented in this exam is mine and mine alone.

_____ _____

Signature (electronic is OK if you do not return a scanned exam) Date

(Unsigned exams will be given a grade of zero.)

1. (15) If a process in an MPI communicator world of size *N* needs to send a message to all other processes in the world, that process can issue *N*-1 `MPI_Send` (or `MPI_Isend`) calls, or it can use `MPI_Bcast` (or `MPI_Ibcast`). There are at least two reasons why either of the broadcast choices is better than either of the send choices. One is obvious at the program source code level in terms of the code you write, the other is due to the specifics of the internal implementation of the broadcast methods. What are these two reasons?

   Source code level: **[Your answer here]**

   Internal implementation: **[Your answer here]**

2. (7) In project 2, there were two types of queries you were to support. For both types of queries, it was important that the rank 0 process use a single collective communication call to transmit required data to all processes in the communicator world.

   To implement those in the first group ("`sr`": scatter-reduce), the rank 0 process was to use `MPI_Scatter` (or `MPI_Iscatter`) to scatter the entire 2D array of data to the other processes in the world. Tasks in this category included finding min, max, and average values found in a single column of the data array. In this case, each rank process handled a portion of the requested single column.

   To implement those in the second group ("`bg`": broadcast-gather), the rank 0 process was to use `MPI_Bcast` (or `MPI_Ibcast`) to broadcast the entire 2D array of data to the other processes in the world. Tasks in this category included finding min, max, and average values found in several columns of the data array. In this case, each rank process was to handle the entirety of one of the requested columns, and hence *N* was required to be the number of columns whose min, max, or average was requested.

   In the project assignment, I posed the question: "*Do you know why I specified "scatter" for the queries in #4, but "broadcast" for the queries in #5? Obviously I want you to get experience with both, but there is more to it than that.*".

   So what was the "more to it than that"?

   [Your answer here]

3. (21) I am implementing an algorithm using OpenMPI that uses two large input data sets and produces an output data set from them. Let's call the input data sets *DS1* and *DS2*; the output data set will be called *DS3* and will be of the same size as *DS2*.

   The rank 0 process is responsible for reading the two input data sets and transmitting relevant portions of them to the other processes in the world. The nature of the algorithm is such that all processes need the entirety of *DS1*, but each process needs only a slice of *DS2*, producing the corresponding slice of the output data set *DS3*.

   The general structure of the algorithm from the perspective of the rank 0 process is:

   1. Read *DS1* and *DS2*.
   2. Transmit *DS1* to all rank processes.
   3. Transmit the relevant portions of *DS2* to the processes in the world. (Assume that all ranks will get subsets of *DS2* of the same size.)
   4. All ranks (including rank 0) perform their task using *DS1* and their assigned portion of *DS2*, generating their portion of the output data set *DS3*.
   5. Rank 0 accepts from the other rank processes their portion of the resulting data set DS3.

   We have studied the following collective communication APIs:

   ```
   MPI_Bcast, MPI_Ibcast, MPI_Scatter, MPI_Iscatter, MPI_Scatterv,
   MPI_Iscatterv, MPI_Gather, MPI_Igather, MPI_Gatherv,
   MPI_Igatherv, MPI_Reduce, MPI_Ireduce, MPI_Alltoall,
   MPI_Allgather, MPI_Allreduce.
   ```

   For each of steps 2, 3, and 5 above, state which of these collective communication calls would be used – including whether the immediate or blocking version is used. And state why.

   Step 2: **[Your answer here]**
   Step 3: **[Your answer here]**
   Step 5: **[Your answer here]**

4. (7) Why is there an `MPI_Gatherv` and an `MPI_Scatterv`, but there is no `MPI_Bcastv`?

   [Your answer here]

5. Both CUDA and OpenCL kernel launches are structured as a two-level hierarchy: (i) a 1D, 2D, or 3D array of thread blocks (CUDA) or Work Groups (OpenCL), and (ii) a 1D, 2D, or 3D array of threads comprising the threads of a single thread block (CUDA) or Work Group (OpenCL). The threads at this second level are further segmented into groups of 32 threads called warps (CUDA) or wavefronts (OpenCL).

   (a) (10) We refer to GPUs as SIMD machines. What does SIMD mean in terms of how the kernels execute, and to what grouping summarized above – (i), (ii), or warp/wavefront – does the term primarily apply?

   [Your answer here]

   (b) (8) The GPU is structured as several (15 is not uncommon) Streaming Multiprocessors (SMs), each with several hundred cores. What level of the groupings summarized above is used by the GPU scheduler when assigning work to the collection of SMs? That is, the top-level GPU scheduler will assign *X* to SMs in a round-robin fashion. Will *X* be an entire computational grid, a thread block (work group), a warp/wavefront, or an individual thread?

   [Your answer here]

   (c) (8) Once *X* has been assigned, can it be migrated to another SM? Bonus (3 extra points): why or why not?

   [Your answer here]

6. (8) What is meant by the term *thread divergence*, and why are we concerned with that in GPU programming?

   [Your answer here]

7. (8) Each thread in a computational grid has both a "local ID" and a "global ID". What is the difference between these two IDs?

   [Your answer here]

8. (8) We observed that the OpenCL data type returned from the OpenCL API call to dynamically allocate memory on the GPU was `cl_mem` rather than an ordinary pointer as the corresponding CUDA calls return. Explain what the cl_mem data type encapsulates with respect to the general computational environment assumed by OpenCL.

   [Your answer here]