EE 451: Parallel and Distributed Computation

PA6a — Spring 2021

Due date: Monday 12th April 2021 11:59 PM

1. **Examples**

   Copy example files to your home directory.

   1. Login to HPC
   2. Copy

      ```
      cp -r /project/xuehaiqi_652/6a .
      ```

   The `hello.cu` contains the CUDA implementation of HelloWorld.

   1. Login to HPC
   2. Setup MPI toolchain:

      ```
      module purge
      module load gcc/8.3.0 cuda/10.1.243
      ```

   3. Compile

      ```
      nvcc -O3 hello.cu
      ```

   4. Run

      ```
      srun -n1 --gres=gpu:1  -t1 ./a.out
      ```

      The option `-t` specifies the limit of run time. Setting it as a small number will get your program scheduled earlier. For more information on `srun` options, you can use `man srun` to find out.

   5. Profile (optional)

      ```
      srun -n1 --gres=gpu:p100:1 --partition=debug nvprof ./a.out
      ```

   6. Allocate a machine

      ```
      salloc -n1 --gres=gpu:1 --mem=16G -t10
      // After the allocation, you will log on the machine and have
         10 minutes to perform multiple operations
      ./a.out
      // edit, compile, and run again without waiting for a new
         allocation
      ./a.out
      ./a.out
      ```

2. (15 points) Refer to the file `globalVaribale.cu`. Declare statically a global float array with a size of five elements. Initialize the global array with the same value of `3.14`. Modify the kernel to let each thread change the value of the array element with the same index as the thread index. Let the value be multiplied with the thread index. Invoke the kernel with five threads.

3. (15 points) Modify `sumArrayZerocpy.cu` to access A, B, and C at an offset. Compare performance with and without L1 cache enabled.

4. (15 points) Modify `readWriteSegment.cu`. Apply an unrolling factor of four to `readWriteOffset` and compare performance with the original.

5. (15 points) Modify `transpose.cu`. Refer to the kernel `transposeUnroll4Row`. Implement a new kernel, `transposeUnroll8Row`, to let each thread handle eight elements. Compare the performance with existing kernels.

6. (15 points) Modify `transpose.cu`. Refer to the kernels `transposeDiagonalCol` and `transposeUnroll4Row`. Implement a new kernel, `transposeDiagonalColUnroll4`, to let each thread handle four elements. Compare the performance with existing kernels.

7. (25 points) A CUDA program has $2K$ threads. 3 GB of data need to be transferred from CPU to GPU and 1 GB of result data need to be transferred back from GPU to CPU. Data transfer is through PCIe whose bandwidth is 16 GB/s. The GPU has $1K$ CUDA cores and can run at most $1K$ threads in parallel. Each CUDA core runs at 1 GHz and is able to perform 1 multiply-add operation in each clock cycle. Each access to global memory takes 100 cycles. Assume each thread needs 10 memory accesses to the global memory and 100 multiply-add operations. What is the execution time of the CUDA program in the best case? Clearly state your assumptions.

   The total execution time consists of three parts: Data transfer latency (CPU to GPU), Kernel execution and Data transfer latency (GPU to CPU).

   Submission Instructions: Submit your code, screenshots, and a performance report as described above.