Advanced Topics in Numerical Analysis: High Performance Computing Assignment 4 (due Apr. 18, 2022)

Handing in your homework: Hand in your homework as for the previous homework assignments (git repo with Makefile), answering the questions by adding a text or a LaTeX file to your repo.

1. **Matrix-vector operations on a GPU.** Write CUDA code for an inner product between two given (long) vectors on a GPU. Then, generalize this code to implement a matrix-vector multiplication (no blocking needed here) on the GPU. Check the correctness of your implementation by performing the same computation on the CPU and compare them. Report the memory band your code obtains on different GPUs.[1]
   Solution:
   Implemented vector vector multiplication in the file: vectorVectorProduct.cu
   Please compile it using:
   nvcc -std=c++11 vectorVectorProduct.cu -o vectorVectorProduct -Xcompiler -fopenmp

   Following are the images with the bandwidth on different gpus.

```
Outer time GPU: 0.001319 s, Inner time GPU: 0.000258
Speed Up Outer: 1.390347
Speed Up Inner: 7.095701
Bandwidth: 31.7953 GB/s
Error = 0.000000
[uk2051@cuda1 hw4]$
```

Figure 1: Courant CUDA Server: 1

```
[uk2051@cuda2 hw4]$ ./vectorVectorProduct
Blocks Count: 256 |Threads per block: 1024 |No of entities in vector: 262144
CPU:  0.001838 s
Outer time GPU: 0.001025 s, Inner time GPU: 0.000131
Speed Up Outer: 1.792513
Speed Up Inner: 14.019103
Bandwidth: 40.9179 GB/s
Error = 0.000000
[uk2051@cuda2 hw4]$
```

Figure 2: Courant CUDA Server: 2

[1] The cuda{1–5}.cims.nyu.edu compute servers at the Institute have different Nvidia GPUs, for an overview see the list of compute servers available at the Institute: https://cims.nyu.edu/webapps/content/systems/resources/computeservers.

```
[uk2051@cuda3 hw4]$ ./vectorVectorProduct
Blocks Count: 256 |Threads per block: 1024 |No of entities in vector: 262144
CPU:  0.002188 s
Outer time GPU: 0.001861 s, Inner time GPU: 0.000863
Speed Up Outer: 1.175754
Speed Up Inner: 2.534911
Bandwidth: 22.5376 GB/s
Error = 0.000000
```

Figure 3: Courant CUDA Server: 3

```
[uk2051@cuda4 hw4]$ ./vectorVectorProduct
Blocks Count: 256 |Threads per block: 1024 |No of entities in vector: 262144
CPU:  0.002000 s
Outer time GPU: 0.001360 s, Inner time GPU: 0.000134
Speed Up Outer: 1.470612
Speed Up Inner: 14.898752
Bandwidth: 30.849 GB/s
Error = 0.000000
[uk2051@cuda4 hw4]$
```

Figure 4: Courant CUDA Server: 4

```
[uk2051@cuda5 hw4]$ ./vectorVectorProduct
Blocks Count: 256 |Threads per block: 1024 |No of entities in vector: 262144
CPU:  0.001790 s
Outer time GPU: 0.002361 s, Inner time GPU: 0.000314
Speed Up Outer: 0.758193
Speed Up Inner: 5.708081
Bandwidth: 17.7671 GB/s
Error = 0.000000
```

Figure 5: Courant CUDA Server: 5

Implemented matrix vector multiplication in the file: matrixVectorProduct.cu
Please compile it using:
nvcc -std=c++11 matrixVectorProduct.cu -o matrixVectorProduct -Xcompiler -fopenmp

Following are the images with the bandwidth on different gpus.

```
[uk2051@cuda1 hw4]$ nvcc -std=c++11  matrixVectorProduct.cu -o matrixVectorProduct -Xcompiler -fopenmp
[uk2051@cuda1 hw4]$ ./matrixVectorProduct
Blocks Count: 8192 |Threads per block: 1024 |No of columns: 4096 |No of rows: 2048
CPU 0.027584 s
Outer time GPU: 0.013556 s, Inner time GPU: 0.001728
Speed Up Outer: 2.034839
Speed Up Inner: 15.958843
Bandwidth: 54.4629 GB/s
Error = 0.000000
[uk2051@cuda1 hw4]$
```

Figure 6: Courant CUDA Server: 1

```
[uk2051@cuda2 hw4]$ ./matrixVectorProduct
Blocks Count: 8192 |Threads per block: 1024 |No of columns: 4096 |No of rows: 2048
CPU 0.029569 s
Outer time GPU: 0.008159 s, Inner time GPU: 0.000882
Speed Up Outer: 3.624038
Speed Up Inner: 33.534027
Bandwidth: 90.4884 GB/s
Error = 0.000000
[uk2051@cuda2 hw4]$
```

Figure 7: Courant CUDA Server: 2

```
Blocks Count: 8192 |Threads per block: 1024 |No of columns: 4096 |No of rows: 2048
CPU 0.041055 s
Outer time GPU: 0.015154 s, Inner time GPU: 0.001947
Speed Up Outer: 2.709092
Speed Up Inner: 21.090843
Bandwidth: 48.7183 GB/s
Error = 0.000000
[uk2051@cuda3 hw4]$
```

Figure 8: Courant CUDA Server: 3

```
[uk2051@cuda4 hw4]$ ./matrixVectorProduct
Blocks Count: 8192 |Threads per block: 1024 |No of columns: 4096 |No of rows: 2048
CPU 0.031762 s
Outer time GPU: 0.015972 s, Inner time GPU: 0.001234
Speed Up Outer: 1.988575
Speed Up Inner: 25.734574
Bandwidth: 46.2234 GB/s
Error = 0.000000
[uk2051@cuda4 hw4]$
```

Figure 9: Courant CUDA Server: 4

```
[uk2051@cuda5 hw4]$ ./matrixVectorProduct
Blocks Count: 8192 |Threads per block: 1024 |No of columns: 4096 |No of rows: 2048
CPU 0.027613 s
Outer time GPU: 0.042590 s, Inner time GPU: 0.002102
Speed Up Outer: 0.648360
Speed Up Inner: 13.135562
Bandwidth: 17.3351 GB/s
Error = 0.000000
[uk2051@cuda5 hw4]$
```

Figure 10: Courant CUDA Server: 5

2. **2D Jacobi method on a GPU.** Implement the 2D Jacobi method as discussed in the 2nd homework assignment using CUDA. Check the correctness of your implementation by performing the same computation on the CPU and compare them.
   Solution:
   implemented 2d jacobi for gpu using cuda. Ran it for 2000 iterations. Computed the maximum relative error with respect to sequential code and computed maximum residual error norm.

```
[uk2051@cuda2 hw4]$ ./jacobi2d
Number of discretization points: 1024
Gird Shape: Blocks in x_dir: 32 Blocks in y_dir: 32 | Block Shape: Threads in x_dir: 32 Threads in y_dir: 32
Iteration: 1999 |Seq relative error: 2.15479e-06  |Residual Error: 3.0848e-07
[uk2051@cuda2 hw4]$
```

Figure 11: Jacobi Output (printing only last iteration)

3. **Update on final projection** Describe with a few sentences the status of your final project: What tasks that you formulated in the proposal have you worked on? Did you run into unforeseen issues?

   Solution:

   Status of the project:

   We are implementing some graph algorithms like Breadth First Search, Strongly connected components, Bellman Ford, etc using parallel programming for shared memory architecture. One important thing to note is that we are not simply using OpenMP for loops for parallelization, but we are using a variant of algorithm especially designed for parallel computation.

   Developing of an interface for processing big graphs and instantiating relevant data structures have taken more than a week. We have successfully implemented infrastructure for parallelizing these graph algorithms and have tested our current development graphs of sizes ranging from 10 nodes to 10000 nodes. We have implemented parallel version of Breadth First Search and now will be testing on big graphs as well as analyzing the results. We ran into some technical issues like how to use C++ atomic class, or how to write a make file for many different c++ files and only one executable.

   One major issue that we encountered is that it would be tough to use GPUs for our algorithms as we don't have much floating point operations.