# Spring 2019: Advanced Topics in Numerical Analysis: High Performance Computing Assignment 4

Kaizhe Wang (kw2223)

1. **Matrix-vector operations on a GPU.**

   First check the correctness of my implementation: run codes with different matrix size, compare the CPU result and GPU result. This process is done on cuda2.cims.nyu.edu server, GeForce RTX 2080 Ti. For vector-vector inner product (vec_vec_mult.cu), if the size of vector is less than $2^{30}$, CPU and GPU results are consistent. For the matrix-vector multiplication (mat_vec_mult.cu), if the size of matrix is less than $2^{16}$, two results are consistent. If the size of matrix is greater than the critical value, the GPU result is different from the CPU result.

   Then I run the CUDA code on different GPUs, to get the performance of each GPU. Bandwidth1 is obtained by vector-vector inner product, the size of vector I used is $2^{28}$; and bandwidth2 is obtained by performing matrix-vector multiplication, where the matrix size I used is $2^{14}$.

   (cuda4 and cuda5 were overload when I tried to run my code, so the performance of these GPUs are not determined by me yet.)

| Hostname | GPU | Bandwidth1 (GB/s) | Bandwidth2 (GB/s) |
|---|---|---|---|
| cuda1 | TITAN Black (6 GB) | 73.311057 | 5.269748 |
| cuda2 | 2080 Ti (11 GB) | 204.219057 | 6.399807 |
| cuda3 | TITAN V (12 GB) | 318.870908 | 9.300983 |
| cuda4 | TITAN X (12 GB) | TBD | TBD |
| cuda5 | TITAN Z (12 GB) | TBD | TBD |

   The bandwidth obtained by vector-vector inner product is much greater then by matrix-vector multiplication, I think it's because in the matrix-vector multiplication, the inner product of each row of matrix and the vector, is done in serial, not parallel.

2. **2D Jacobi method on a GPU.**

   To get the performance of different GPUs, I run my codes with matrix size $1024 \times 1024$, maximum iteration number is 10000, report the running time on GPUs.

| Hostname | GPU | Time (s) |
|---|---|---|
| cuda1 | TITAN Black (6 GB) | 6.590403 |
| cuda2 | 2080 Ti (11 GB) | 2.457543 |
| cuda3 | TITAN V (12 GB) | 1.146881 |
| cuda4 | TITAN X (12 GB) | TBD |
| cuda5 | TITAN Z (12 GB) | TBD |

3. **Pitch your final project.**

   Topic: Correlation function with jackknife.

   Teammate: Yucheng Zhang.

   General Introduction: For this final project, we will do the estimation of correlation function of galaxies, which is one of the most important applications of High Performance Computing in Cosmology. The most computation intensive part is to calculate the distance between two sets of points and fit them into proper bins.

   The things we plan to do include:

   (a) Figure out the mesh algorithm. Naive pair count requires $O(N^2)$ time complexity, where N is the size of the data. With the mesh algorithm, we hope to reduce the time complexity to $O(NlogN)$.

   (b) Figure out the way to jackknife resampling in the code. The input data points are divided into $njk$ parts, where $njk$ stands for the number of jackknife regions.

   (c) Parallelize the code with OpenMP and MPI in C.

   (d) Wrap the parallelized C with Python, which works better in loading data and plotting results.