HPC Homework IV

Zihan Zhang

April 2022

I use the CIMS GPU Computing - NVIDIA/CUDA machines to implement this assignment. The modules environment is gcc-4.9.2 and CUDA-9.2. Also, I found Professor Georg Stadler's former material of this course is very inspiring, which is related to Professor Peherstorfer's code. I used it as a reference. The configurations of these servers are listed[1]:

Hostname	GPU
cuda1	GeForce GTX TITAN Black (6 GB memory)
cuda2	GeForce RTX 2080 Ti (11 GB memory)
cuda3	TITAN V (12 GB memory)
cuda4	GeForce GTX TITAN X (12 GB memory)
cuda5	GeForce GTX TITAN Z (12 GB memory)

Problem 1.1. The source code is **vec-vec-ip.cu**. I set the length of vectors as $n = 2^{20}$ and test the results on different GPUs. The CPU and GPU results are always consistent, if the size of vector is $\leq 2^{28}$.

```
cuda1
CPU Bandwidth = 24.270767 GB/s
GPU Bandwidth = 27.850256 GB/s
Error = 0.000000

cuda2
CPU Bandwidth = 1.065149 GB/s
GPU Bandwidth = 54.166526 GB/s
Error = 0.000000

cuda3
CPU Bandwidth = 0.324396 GB/s
GPU Bandwidth = 71.909532 GB/s
Error = 0.000000

cuda4
CPU Bandwidth = 0.324772 GB/s
```

[1] https://cims.nyu.edu/webapps/content/systems/resources/computeservers

```
GPU Bandwidth = 39.889144 GB/s
Error = 0.000000

cuda5
CPU Bandwidth = 20.117917 GB/s
GPU Bandwidth = 23.854315 GB/s
Error = 0.000000
```

Problem 1.2. The source code is in **matrix.cu**, which considers a $M \times N$ matrix multiplied by a $N \times 1$ vector. To generate a more comparable result, we make $M = N = 2^{10}$.

```
cuda1
CPU Bandwidth = 4.905224 GB/s
GPU Bandwidth = 0.481354 GB/s
Error = 0.000000

cuda2
CPU Bandwidth = 4.962704 GB/s
GPU Bandwidth = 0.554842 GB/s
Error = 0.000000

cuda3
CPU Bandwidth = 1.285991 GB/s
GPU Bandwidth = 0.647302 GB/s
Error = 0.000000

cuda4
CPU Bandwidth = 0.645636 GB/s
GPU Bandwidth = 0.528436 GB/s
Error = 0.000000

cuda5
CPU Bandwidth = 51.531032 GB/s
GPU Bandwidth = 0.341578 GB/s
Error = 0.000000
```

Noteworthy, the bandwidth in the vector-vector inner product is much greater than the matrix-vector multiplication, because in the latter case, the inner product of each row of matrix with the vector is done serially not in parallel.

Problem 2.2. The source code is in **jacobi-cuda.cu**. The difference of CPU-GPU result is around -4.7490810^{-16}, which is negligible. For the CPU omp code, I use 4 threads.

```
cuda1
========CPU========
use threads: 4
```

```
Initial Residual:20
Remaining res:9.92486e-09
CPU Bandwidth = 58.287557 GB/s
CPU Time:0.0664454s
========GPU========
Initial Residual:20
Remaining res:9.92486e-09
GPU Bandwidth = 20.864774 GB/s
GPU Time:0.185611s
========Result Comparison========
CPU-GPU difference:-4.74908e-16

cuda2
========CPU========
use threads: 4
Initial Residual:20
Remaining res:9.92486e-09
CPU Bandwidth = 54.100262 GB/s
CPU Time:0.0715889s
========GPU========
Initial Residual:20
Remaining res:9.92486e-09
GPU Bandwidth = 11.701614 GB/s
GPU Time:0.330911s
========Result Comparison========
CPU-GPU difference:-4.74908e-16

cuda3
========CPU========
use threads: 4
Initial Residual:20
Remaining res:9.92486e-09
CPU Bandwidth = 39.437915 GB/s
CPU Time:0.0981944s
========GPU========
Initial Residual:20
Remaining res:9.92486e-09
GPU Bandwidth = 6.055175 GB/s
GPU Time:0.639463s
========Result Comparison========
CPU-GPU difference:-4.74908e-16

cuda4
========CPU========
use threads: 4
```

```
Initial Residual:20
Remaining res:9.92486e-09
CPU Bandwidth = 39.759350 GB/s
CPU Time:0.0974018s
========GPU========
Initial Residual:20
Remaining res:9.92486e-09
GPU Bandwidth = 11.026700 GB/s
GPU Time:0.351165s
========Result Comparison========
CPU-GPU difference:-4.74908e-16

cuda5
========CPU========
use threads: 4
Initial Residual:20
Remaining res:9.92953e-09
CPU Bandwidth = 52.446678 GB/s
CPU Time:0.0738446s
========GPU========
Initial Residual:20
Remaining res:9.92486e-09
GPU Bandwidth = 11.846828 GB/s
GPU Time:0.326856s
========Result Comparison========
CPU-GPU difference:4.67203e-12
```

Problem 3. Final Project: Parallel Computation of Fast Fourier Transform. Working with Hongleng Fu and Sixiu Liu.

In the CUDA part, we have written the main function and successfully ran cufft. We have also informed ourselves about the mathematical knowledge of the radix-2, 4, and 8 algorithms. Our experiment design is to implement the radix-2 FFT and compare it with the cufft library, running parallel on the CIMS CUDA servers. Similarly, for OpenMP, we will also use the library as a benchmark and implement the corresponding OpenMP version. We are currently working on transferring the cfft2 and step function mentioned in Peterson's book to the C++ version. Our implementation is non-recursive and is based on the conventional Cooley-Tukey algorithm.

In the presentation and report, we will go through the algorithms of FFT, especially why the CUDA and OpenMP implementations can expedite the algorithms.

Our GitHub repository: https://github.com/StevenZhang0116/HPC-FFT.git