Programming Project #2 Report

Rushikesh Khamkar

<u>Steps Used For Excution</u>

With my eRaider ID and password, I entered the system and made a connection to the HPCC by doing the following actions:

1. Log in into the system using eRaider ID and password
   ssh -J rkhamkar@ssh.ttu.edu rkhamkar@login.hpcc.ttu.edu
2. Clone code from the given github link
   git clone https://github.com/mertside/CS5375_GPU_Lecture
3. cd /CS5375_GPU_Lecture/Project
4. Go to the folder and query to connect to the gpu -> interactive -p gpu-build
5. Part 1 –
   1. Compile single c++ program for matrix multiplication using command - g++ matrixMul_cpu.cpp -o matrixMul_cpu.exe and run using command - ./matrixMul_cpu.exe
   2. Compile parallel program cuda code for matrix multiplication using command - nvcc matrixMul_gpu.cu -o matrixMul_gpu.exe and run using command - ./matrixMul_gpu.exe
   3. Profile this version of your code using a command - nvprof ./matrixMul_gpu_v2.exe
6. Part 2 –
   1. Compile parallel program cuda code for matrix multiplication using command - nvcc matrixMul_gpu_v2.cu -o matrixMul_gpu_v2.exe and run using command - ./matrixMul_gpu_v2.exe
   2. Profile this version of your code using a command - nvprof ./matrixMul_gpu_v2.exe
7. Part 3 –
   1. Compile parallel program cuda code for matrix multiplication using command - nvcc matrixMul_gpu_v3.cu -o matrixMul_gpu_v3.exe and run using command - ./matrixMul_gpu_v3.exe
   2. Profile this version of your code using a command - nvprof ./matrixMul_gpu_v3.exe
8. Part 4 –
   1. Compile parallel program cuda code for matrix multiplication using command - nvcc matrixMul_gpu_v3.cu -o matrixMul_gpu_v3.exe and run using command - ./matrixMul_gpu_v3.exe
   2. Profile this version of your code using a command - nvprof ./matrixMul_gpu_v3.exe

Part 1 – Getting Started - Execute CPU & GPU File

The main difference between CPU and GPU architecture is that a GPU is made to accomplish a broad variety of operations quickly while a CPU is limited in the amount of concurrent tasks it can perform (as shown by CPU clock speed). A GPU is designed to quickly and concurrently render high-resolution images and video.

Below is the difference for CPU and GPU-

CPU -

- Central Processing Unit
- Several cores
- Low latency
- Good for serial processing
- Can do a handful of operations at once

GPU -

- Graphics Processing Unit
- Many cores
- High throughput
- Good for parallel processing
- Can do thousands of operations at once

1. How do these two versions of the code compare? Report how long it takes to execute each version of the matrix multiplication

   In CPU Code, For multiplication, we are using 3 loops for calculation which may cause delay in execution. GPU is introduced to reduce the time for our multiplication. But in our case, we haven't fully handled GPU Cuda version and haven't achieved any parallelisation. We just initialized memory for the variable and at the end, we just remove their allocation using Free function. So, It might happen that we get delayed for GPU Cuda version of the code.

Sr. No.	Size Of Matrix	CPU Version of Multiplication	GPU Version of the Multiplication
1	512 X 512	0.003	0.037

2. Is the GPU version optimal? Report why or why not?

   GPU version is not optimal here. Because, we haven't made any changes for loops. We just have made changes for allocation and deallocation of the variables. In this, we have to make changes for improving the loops. And we need to consider threadID and blockDIm to improve the performance. In CPU Code, For multiplication, we are using 3 loops for calculation which may cause delay in execution. GPU is introduced to reduce the time for our multiplication. But in our case, we haven't fully handled GPU Cuda version and haven't achieved any parallelisation.

Part 2 – Multiple Threads

Every thread in CUDA has a specific index assigned to it that allows it to calculate and access certain memory addresses in an array. Think of a scenario where an array contains 512 elements. A grid with a single block that has 512 threads is one of the organizational structures.

Sr. No.	Size Of Matrix	GPU Part 1	GPU Part 2
1	512 X 512	0.037	0.009333

Part 3 – Multiple Blocks

If you wait for an update of a block that hasn't been scheduled, block scheduling may become deadlocked. The only method to avoid this deadlock since CUDA cannot ensure that scheduled blocks run in a specified order is to set a grid size restriction that allows all blocks to run concurrently. Assume that each SM on a CUDA device is limited to a maximum of 1,024 threads or 8 blocks, whichever comes first. Furthermore, each block may have up to 512 threads.

For this, we have used threadIdx and gridDim for calculation –

```
int i = (gridDim.x*gridDim.y)*blockIdx.z+(blockIdx.y*gridDim.x)+(blockIdx.x);
int j = (blockDim.x*blockDim.y)*threadIdx.z+(threadIdx.y*blockDim.x)+(threadIdx.x);
int T = blockDim.x*blockDim.y*blockDim.z;
int B = gridDim.x*gridDim.y*gridDim.z;
```

We have parallelize our ith and jth loop to achieve better performance by calculation with above values.

Sr. No.	Size Of Matrix	GPU Part 2	GPU Part 3
1	512 X 512	0.009333	0.01

Part 4 – Optimize

1. **Can you optimize the number of threads and blocks you use? Report your effort**

   Since shared memory is a scarce resource and kernels frequently have very particular requirements that limit the numerous factors affecting parallelism, you might wish to take this into account first when using shared memory. Either blocks with a big number of threads sharing a vast region, or blocks with a small number of threads sharing a small region (under constant occupancy). We have substitute our values in other cuda kernel version using cudamemalloc. And , then. In runtime we can fetch those values from host to device. To allocate memory in host will be helpful for us to achieve parallelism in runtime.

2. **Move the data initialization to the GPU in another CUDA kernel and prefetch the data to GPU memory before running the kernel. Did the page faults decrease? Report why or why not?**

In Our case, we have achieved the initialization by moving values from Device to Host, and in runtime, we are fetching those values in Device again for multiplication. It won't impact much in our scenario, as we have less data, so it won't be much impactful in our case. As we got run time 0.01900 in our previous file and after fix, we got 0.30800 which is more than. It might be because of the cuda execution.