COA Programming Project 2

Sravani Manduva – R11800063

PART 1:

1. In the file matrix_Mul_gpu_part1.cu did changes related to the GPU version and executed using the commands nvcc matrix_Mul_gpu_part1.cu -o matrix_Mul_gpu_part1.exe but written a Makefile by using **make** command all 5 files matrixMul_cpu.cpp, matrix_Mul_gpu_part1, matrix_Mul_gpu_part2, matrix_Mul_gpu_part3, matrix_Mul_gpu_part4 and by using **clean** command all generated .exe will be removed.
2. For the CPP execution it is normal Matrix Multiplication, In GPU execution first we need to assign the memory in the CPU and take the execution to the GPU. We use the cudaMallocManaged() method to allocate the memory and cudaFree() method to free the memory. The rest of the code and execution is the same, since we call using the Kernel configuration GPUmatmul<<<1,1>>> which is no different from the cpp execution. CPU execution takes around 851.7200 ms while the GPU takes 9408.104 ms.

   Below images are results of both CPU and GPU version of code

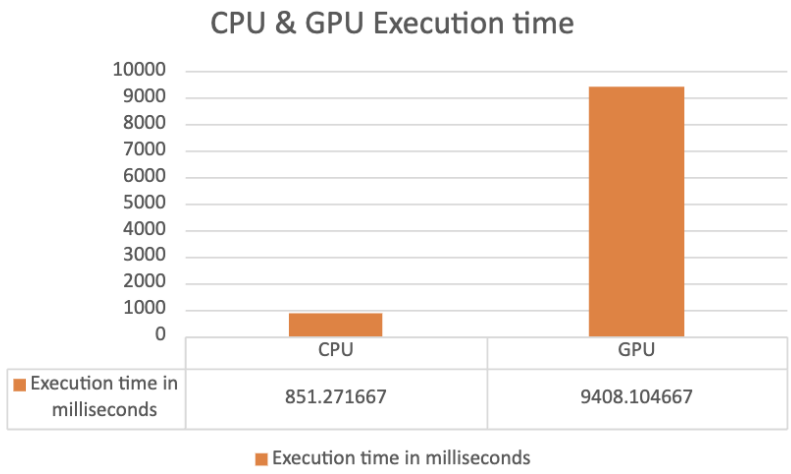
```
login-20-25:/COA_Project2$ ls
matrixMul_cpu.cpp  matrixMul_cpu.ext  matrixMul_gpu.cu  matrix_Mul_gpu_part1.cu
matrix_Mul_gpu_part2.cu  matrix_Mul_gpu_part3.cu  matrix_Mul_gpu_part4.cu  README.
md  results
login-20-25:/COA_Project2$ g++ matrixMul_cpu.cpp -o matrixMul_cpu.ext
login-20-25:/COA_Project2$ ./matrixMul_cpu.ext
Size of matrix (N) is 512 by 512.
Starting CPU computation
It took 851.271667 ms on avg.
RUN OK.
login-20-25:/COA_Project2$ time ./matrixMul_cpu.ext
Size of matrix (N) is 512 by 512.
Starting CPU computation
It took 851.720000 ms on avg.
RUN OK.

real    0m3.986s
user    0m3.971s
sys 0m0.003s
login-20-25:/COA_Project2$
```

```
gpu-20-11:/COA_Project2$ nvprof ./matrix_Mul_gpu_part1.ext
Size of matrix (N) is 512 by 512.
==17954== NVPROF is profiling process 17954, command: ./matrix_Mul_gpu_part1.ext
Starting unoptimized GPU computation
It took 23197.545333 ms on avg.
RUN OK.
==17954== Profiling application: ./matrix_Mul_gpu_part1.ext
==17954== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  82.0224s         4  20.5056s  12.3108s  23.2619s  GPUmatmul(int, double*, double*, double*)
      API calls:   99.74%  82.0289s         4  20.5072s  12.3108s  23.2641s  cudaDeviceSynchronize
                    0.26%  211.57ms         3  70.522ms  28.684us  211.48ms  cudaMallocManaged
                    0.00%  794.03us         3  264.68us  162.41us  442.51us  cudaFree
                    0.00%  566.78us         1  566.78us  566.78us  566.78us  cuDeviceTotalMem
                    0.00%  287.03us         4  71.757us  55.866us  114.84us  cudaLaunchKernel
                    0.00%  95.681us       101     947ns     124ns  38.646us  cuDeviceGetAttribute
                    0.00%  17.346us         1  17.346us  17.346us  17.346us  cuDeviceGetName
                    0.00%  4.5660us         1  4.5660us  4.5660us  4.5660us  cuDeviceGetPCIBusId
                    0.00%     859ns         3     286ns     118ns     548ns  cuDeviceGetCount
                    0.00%     512ns         2     256ns     147ns     365ns  cuDeviceGet
                    0.00%     291ns         1     291ns     291ns     291ns  cuDeviceGetUuid

==17954== Unified Memory profiling result:
Device "Tesla V100-PCIE-32GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      36  170.67KB  4.0000KB  0.9961MB  6.000000MB  591.1040us  Host To Device
      12  170.67KB  4.0000KB  0.9961MB  2.000000MB  171.8400us  Device To Host
      15         -         -         -           -  2.150112ms  Gpu page fault groups
Total CPU Page faults: 24
```

Below chart that represents the Execution time of both CPU and GPU



3. since the memory should be made at the CPU, and the interaction needs to allocate a resource at the GPU level and afterward it should be executed. In this way, it requires some time to execute at GPU, since no improvement is made in GPU Level because if we see the results GPU takes more time to execute than the CPU.

PART 2. MULTIPLE THREADS

1. In this Part 2 executed with few changes in matrix_Mul_gpu_part2.cu file related to the Multiple threads. Part2 Execution took around 275.9063 ms it is less than Part 1 execution. Here total number of page faults for both Part 1 and Part 2 is 24.

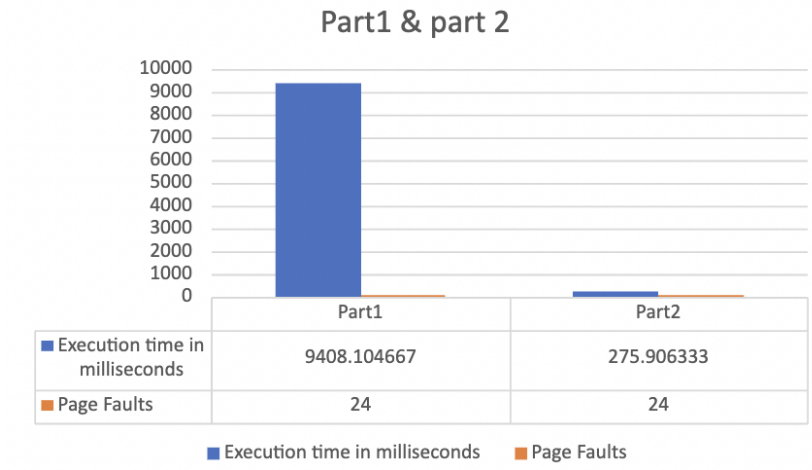
2. In Part1 GPU code we just use the cudaMallocManaged() method to allocate the memory and cudaFree() method to free the memory, but in part 2, we have utilized the threadIdx and

blockDim here in this block dimensions are used for the strides which the iteration over the blocks goes through strides. Below is a time comparison graph for Part 1 and Part 2 for detailed execution times that I generated the ASCII file in the code that is present in the results folder.

```
gpu-20-11:/COA_Project2$ nvprof ./matrix_Mul_gpu_part2.ext
Size of matrix (N) is 512 by 512.
==18392== NVPROF is profiling process 18392, command: ./matrix_Mul_gpu_part2.ext
Starting unoptimized GPU computation
It took 275.830667 ms on avg.
RUN OK.
==18392== Profiling application: ./matrix_Mul_gpu_part2.ext
==18392== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  1.11498s         4  278.75ms  276.21ms  286.27ms  GPUmatmul(int, double*, double*, double*)
      API calls:   78.70%  1.11501s         4  278.75ms  276.22ms  286.27ms  cudaDeviceSynchronize
                   21.21%  300.50ms         3  100.17ms  30.338us  300.41ms  cudaMallocManaged
                    0.04%  573.99us         1  573.99us  573.99us  573.99us  cuDeviceTotalMem
                    0.04%  532.07us         3  177.36us  128.80us  247.23us  cudaFree
                    0.01%  102.21us         4  25.553us  6.9900us  48.373us  cudaLaunchKernel
                    0.01%  94.321us       101     933ns     116ns  39.362us  cuDeviceGetAttribute
                    0.00%  17.764us         1  17.764us  17.764us  17.764us  cuDeviceGetName
                    0.00%  2.2250us         1  2.2250us  2.2250us  2.2250us  cuDeviceGetPCIBusId
                    0.00%  1.0170us         3     339ns     188ns     640ns  cuDeviceGetCount
                    0.00%     840ns         2     420ns     184ns     656ns  cuDeviceGet
                    0.00%     242ns         1     242ns     242ns     242ns  cuDeviceGetUuid

==18392== Unified Memory profiling result:
Device "Tesla V100-PCIE-32GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      42  146.29KB  4.0000KB  0.9961MB  6.000000MB  613.5360us  Host To Device
      12  170.67KB  4.0000KB  0.9961MB  2.000000MB  170.7200us  Device To Host
      17         -         -         -           -  1.858944ms  Gpu page fault groups
Total CPU Page faults: 24
```

Below chart that represents the Execution time and Page faults of Part1 and Part2.



PART 3. MULTIPLE BLOCKS

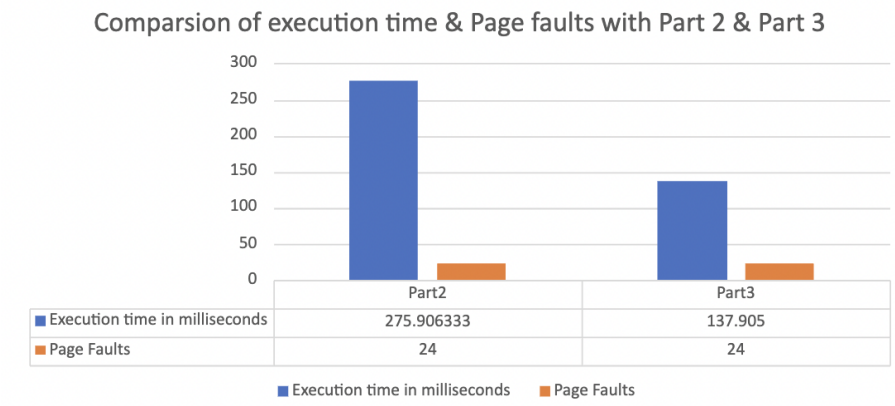
1. Each cell in the matrix should be assigned to a different thread. Each thread should do O (N * number of assigned cell) computation. Assigned cells of different threads do not overlap with each other. And so, no need for synchronization. Please compile and run your modified code and report the outputs. As per the requirements I have created the blocks and for each block we have assigned 256 threads and calculating no.of blocks using this formula (N+BS-1)/BS by this we can execute the code parallelly GPUmatmul<<<nb,bs>>> (N, x, y, ans). Also consider, there will be no need for synchronization, thus reducing its synchronization time.

2. In this part 3 version of code, we have added different threads to each cell in the matrix by calculating the number of blocks and by default for each block 256 threads are assigned. Here the time taken is 137.9050 ms which is less compared to Part1, and Page fault remains same 24. Still Code is not optimized in this. Below image is the result of Part3 code execution. For detail profiling all the results are added in Results folder in an ASCII format.

```
gpu-20-11:/CUA_Project2$ nvprof ./matrix_Mul_gpu_part3.ext
Size of matrix (N) is 512 by 512.
==19175== NVPROF is profiling process 19175, command: ./matrix_Mul_gpu_part3.ext
Starting unoptimized GPU computation
It took 137.915667 ms on avg.
RUN OK.
==19175== Profiling application: ./matrix_Mul_gpu_part3.ext
==19175== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  554.13ms         4  138.53ms  138.11ms  139.67ms  GPUmatmul(int, double*, double*, double*)
      API calls:   72.43%  554.14ms         4  138.53ms  138.12ms  139.67ms  cudaDeviceSynchronize
                   27.35%  209.27ms         3  69.756ms  9.9870us  209.23ms  cudaMallocManaged
                    0.08%  620.91us         1  620.91us  620.91us  620.91us  cuDeviceTotalMem
                    0.07%  543.19us         3  181.06us  156.84us  214.26us  cudaFree
                    0.06%  420.92us       101  4.1670us     128ns  342.55us  cuDeviceGetAttribute
                    0.01%  58.584us         4  14.646us  6.1340us  29.499us  cudaLaunchKernel
                    0.00%  24.012us         1  24.012us  24.012us  24.012us  cuDeviceGetName
                    0.00%  1.9670us         1  1.9670us  1.9670us  1.9670us  cuDeviceGetPCIBusId
                    0.00%  1.3300us         3     443ns     197ns     919ns  cuDeviceGetCount
                    0.00%     698ns         2     349ns     166ns     532ns  cuDeviceGet
                    0.00%     245ns         1     245ns     245ns     245ns  cuDeviceGetUuid

==19175== Unified Memory profiling result:
Device "Tesla V100-PCIE-32GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      42  146.29KB  4.0000KB  0.9961MB  6.000000MB  611.7440us  Host To Device
      12  170.67KB  4.0000KB  0.9961MB  2.000000MB  169.2800us  Device To Host
      13         -         -         -           -  1.561088ms  Gpu page fault groups
Total CPU Page faults: 24
```

Below chart that represents the Execution time and Page faults of Part2 and Part3.



PART 4. OPTIMIZE

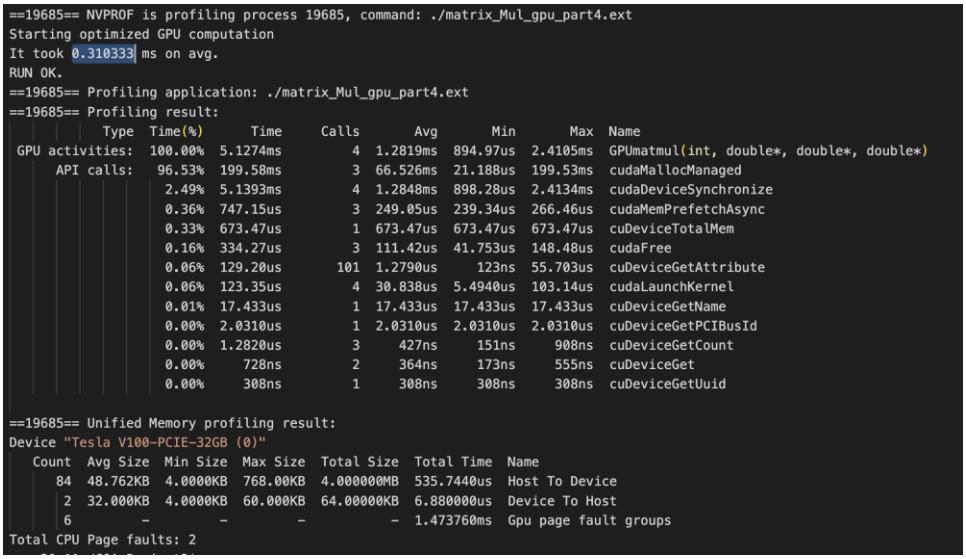
1. Can you optimize the number of threads and blocks you use? Report your effort.

In part 4 execution time is 0.31033 ms and Page faults are 2 Where In this part, the block size is considered 16 and the grid size is (int)ceil (N / block Size) which will reduces the migration overhead. Where in this we can optimize the code to 2d threads and blocks where it is considered into rows and columns at initialization and modified the matrix multiplication code.

2. The data initialization has been moved to CUDA kernel and fetching the data before kernel running. After doing this time has been reduced because we are not dependent upon CPU so here Page faults have been decreased to 2 from 24.

GPU are still significantly more expensive than CPUs and GPU can handle large amounts of parallel computing and data throughput, they struggle when the processing requirements become more chaotic.

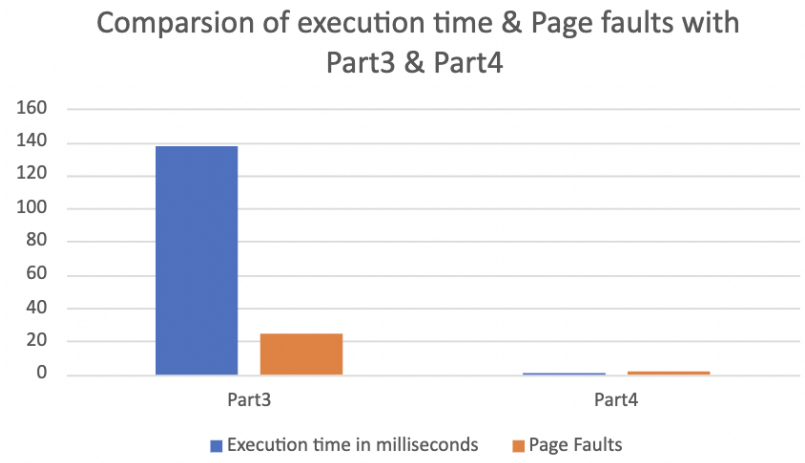
Below image is the result of Part 4 code execution. For detail profiling all the results are added in Results folder in an ASCII format.



```
==19685== NVPROF is profiling process 19685, command: ./matrix_Mul_gpu_part4.ext
Starting optimized GPU computation
It took 0.310333 ms on avg.
RUN OK.
==19685== Profiling application: ./matrix_Mul_gpu_part4.ext
==19685== Profiling result:
            Type  Time(%)      Time     Calls       Avg       Min       Max  Name
 GPU activities:  100.00%  5.1274ms         4  1.2819ms  894.97us  2.4105ms  GPUmatmul(int, double*, double*, double*)
      API calls:   96.53%  199.58ms         3  66.526ms  21.188us  199.53ms  cudaMallocManaged
                    2.49%  5.1393ms         4  1.2848ms  898.28us  2.4134ms  cudaDeviceSynchronize
                    0.36%  747.15us         3  249.05us  239.34us  266.46us  cudaMemPrefetchAsync
                    0.33%  673.47us         1  673.47us  673.47us  673.47us  cuDeviceTotalMem
                    0.16%  334.27us         3  111.42us  41.753us  148.48us  cudaFree
                    0.06%  129.20us       101  1.2790us     123ns  55.703us  cuDeviceGetAttribute
                    0.06%  123.35us         4  30.838us  5.4940us  103.14us  cudaLaunchKernel
                    0.01%  17.433us         1  17.433us  17.433us  17.433us  cuDeviceGetName
                    0.00%  2.0310us         1  2.0310us  2.0310us  2.0310us  cuDeviceGetPCIBusId
                    0.00%  1.2820us         3     427ns     151ns     908ns  cuDeviceGetCount
                    0.00%     728ns         2     364ns     173ns     555ns  cuDeviceGet
                    0.00%     308ns         1     308ns     308ns     308ns  cuDeviceGetUuid

==19685== Unified Memory profiling result:
Device "Tesla V100-PCIE-32GB (0)"
   Count  Avg Size  Min Size  Max Size  Total Size  Total Time  Name
      84  48.762KB  4.0000KB  768.00KB  4.000000MB  535.7440us  Host To Device
       2  32.000KB  4.0000KB  60.000KB  64.00000KB  6.880000us  Device To Host
       6         -         -         -           -  1.473760ms  Gpu page fault groups
Total CPU Page faults: 2
```

Below chart that represents the Execution time and Page faults of Part3 and Part4.



OVERALL RESULTS:

The CPU execution time is 851.720 ms and In GPU Part 1 execution time is more than CPU later it's got decreased in further parts and finally in Part 4 execution time is recorded as 0.310 ms by decreasing the

page faults and changes did changes regarding the threads, Blocks and data initialization have been moved to the GPU in another CUDA kernel and prefetching the data to GPU memory before running the kernel.