BLG562E – Parallel Computing for GPUs using CUDA

Homework #3

Mustafa İzzet Muştu - 504211564

Development and Test Environment

Operating system on the host machine is Windows 11 22H2 and nvcc version is V12.1.66. Hardware information as follows: AMD Ryzen 7 5800X, 32 GB 3400 MHz, MSI RTX 3060 12 GB.

1. Dense Matrix-Matrix Multiplication using GPUs

a) Measure the running time of your kernel with a matrix size of N=1024x1024, M=1024x1024.

I set block dimension as 32x32 (maximum number of threads per block) and utilized CUDA unified memory model. At first, I created 3 matrices with cudaMallocManaged() function. I filled the first two between 0 and 100 on the host code. Then, I initialized the third one with an initializer kernel to improve the matrix multiplication kernel performance by removing the overhead of copying. (Only 1 matrix copy operation is removed, not all of them). To measure the running time of the kernel, I used clock() function from time header and set the precision as 5. I also measured the initilazion times on the host and the GPU. Results can be seen in Table 1.

Trial/Run Time (s)	2 Matrices Initialization on Host	1 Matrix Initialization on GPU	Matrix Multiplication Kernel on GPU	Matrix Multiplication on CPU (1 thread)
1	0.073	0.005	0.004	3.423
2	0.074	0.005	0.004	3.495
3	0.073	0.005	0.004	3.491
4	0.073	0.005	0.003	3.48
5	0.072	0.005	0.005	3.552

Table 1

b) Do a performance profiling using NVVP/nvprof/NSight profiling tool and discuss the performance bottlenecks and identify opportunities for performance optimizations. If the occupancy is a problem work on the Block size to make sure you have enough occupancy per SM, then re-do the profiling.

I profiled the application using Nsight Compute and checked the report. For the matrix multiplication kernel, I got 91.2% SM and memory througput. The plot can be seen in Figure 1.

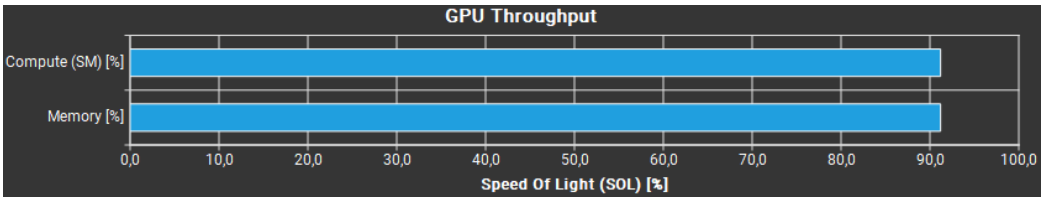

Figure 1

For the occupancy, I got achieved lower result than the theoretical limit. The result is shown in Figure 2. Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. To increase this value, we can decrease the register and shared memory usage in threads and adjust the block size. Since we do not use any shared memory in the kernel and we use only 4 registers per thread that must be used, we need to change the block size. (We can also unsderstand this from the Figure 2, profiler gives us a warning that says occupancy is limited by the number of required registers. We can simply reduce the block size and thus number of required register per block and per SM decreases.)

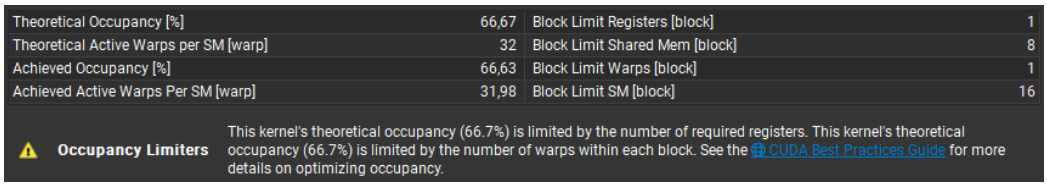

Figure 2

After changing block size from 32x32 to 16x16, occupancy is increased up to 98.39% as expected with a theoretical limit of 100%. SM throughput is also increased up to 97.29%. Results can be seen in Figure 3 and Figure 4.

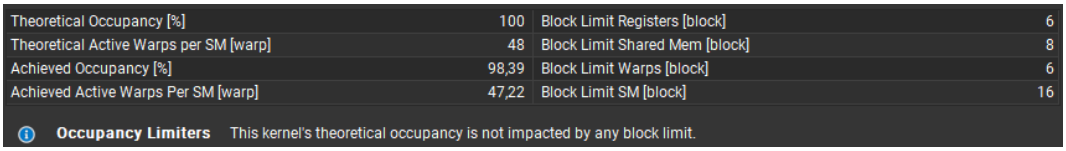

Figure 3

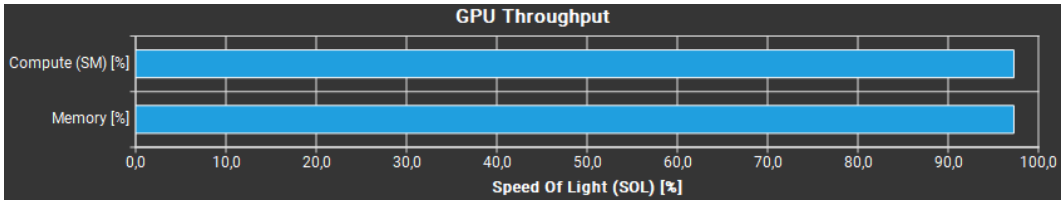

Figure 4

c) <u>Repeat the steps in (a) and (b) with matrix sizes of N=2048x2048, M=2048x2048 and N=4096x4096, M=4096x4096</u>

For a fair comparison, I set the block size as 32x32 at first. Result can be seen in Table 2 and Table 3.

Trial/Run Time (s)	2 Matrices Initialization on Host	1 Matrix Initialization on GPU	Matrix Multiplication Kernel on GPU	Matrix Multiplication on CPU (1 thread)
1	0.289	0.019	0.035	31.875
2	0.296	0.019	0.032	31.646
3	0.291	0.019	0.032	31.716
4	0.293	0.019	0.034	31.749
5	0.296	0.019	0.029	32.355

Table 2- N=M=2048x2048

Trial/Run Time (s)	2 Matrices Initialization on Host	1 Matrix Initialization on GPU	Matrix Multiplication Kernel on GPU	Matrix Multiplication on CPU (1 thread)
1	1.153	0.074	0.197	256.117
2	1.155	0.073	0.197	261.393
3	1.17	0.077	0.219	254.185
4	1.166	0.073	0.198	259.628
5	1.164	0.075	0.197	254.934

Table 3 – N=M=4096x4096

For N=M=2048x2048 and block size as 32x32, I had 66.67% theoretical limit. Profiler warning was the same as part a. So, I changed the block size to 16x16 and occupancy is increased. Before and after values are shown in Figure 5 and Figure 6 respectively. Throughputs are also similar to part a and screenshots are not included.

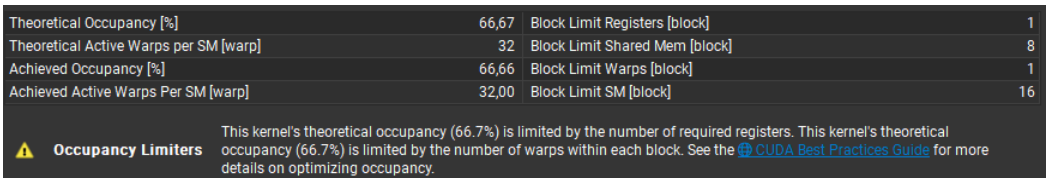

Figure 5

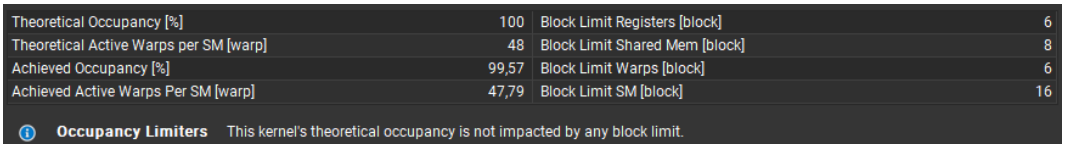

Figure 6

For N=M=4096x4096 I set block size as 16x16 by using the previous knowledge I gained and I got the results that can be seen is Figure 7 and Figure 8.

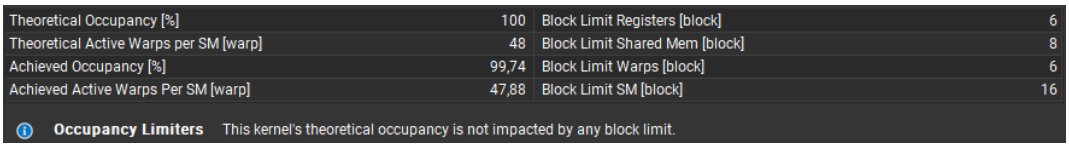

Figure 7

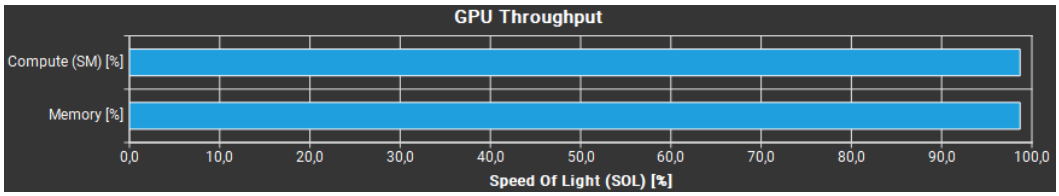

Figure 8

2. Sparse Matrix-Matrix Multiplication using GPUs

a) Measure the running time of your kernel.

I split the sparse matrix-matrix multiplication kernel into 2 part. In the first part, I wrote a kernel in which each threat calculated an output matrix multiplying a non zero value from first matrix and all nonzeros from other. For the second part, I added all those matrices by another kernel. I *chose 685_bus.mtx,685_bus.mtx and 1138_bus.mtx* files from the given link. For all multiplication related kernels, I set block dimension as (16,1,1) and grid dimension was dependent to the input matrix size. I measured the kernel run times for different dimensions, however, I could not measure the time for the last matrix due to lack of memory. Run times of the kernels can be seen in Table 4.

N,M/Run Time (s)	Sparse Matrix Multiplication on CPU (1 thread)	1 Matrix Initialization on GPU	Sparse Matrix Multiplication on GPU (First Part)	Sparse Matrix Multiplication on GPU (Second Part)
494	0.043	0	0	0.004
685	0.186	0	0.001	0.021
1138	-	-	-	-

Table 4

b) Do a performance profiling using NVVP/nvprof/NSight profiling tool and discuss the performance bottlenecks and identify opportunities for performance optimizations. If the occupancy is a problem work on the Block size to make sure you have enough occupancy per SM, then re-do the profiling.

Profiling results can be seen in Figure 9 and Figure 10 for the first part and Figure 11 and 12 for the second part of the sparse matrix-matrix multiplication.

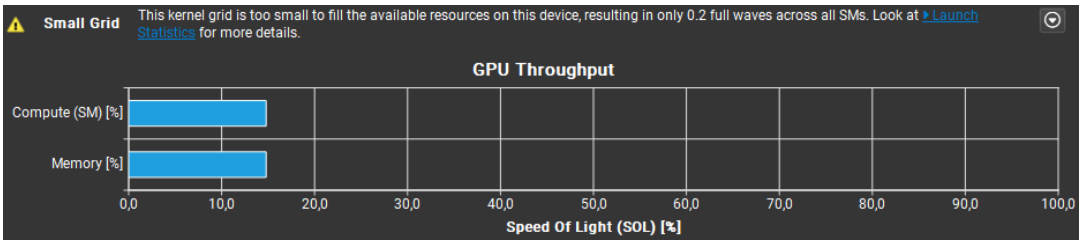

Figure 9

Theoretical Occupancy [%]	33,33	Block Limit Registers [block]	84
Theoretical Active Warps per SM [warp]	16	Block Limit Shared Mem [block]	16
Achieved Occupancy [%]	4,98	Block Limit Warps [block]	48
Achieved Active Warps Per SM [warp]	2,39	Block Limit SM [block]	16

Occupancy Limiters This kernel's theoretical occupancy (33.3%) is limited by the number of blocks that can fit on the SM. This kernel's theoretical occupancy (33.3%) is limited by the required amount of shared memory. The difference between calculated theoretical (33.3%) and measured achieved occupancy (5.0%) can be the result of warp scheduling overheads or workload imbalances during the kernel execution. Load imbalances can occur between warps within a block as well as across blocks of the same kernel. See the CUDA Best Practices Guide for more details on optimizing occupancy.

Impact of Varying Register Count Per Thread

Figure 10

High Memory Throughput Memory is more heavily utilized than Compute: Look at the Memory Workload Analysis section to identify the DRAM bottleneck. Check memory replay (coalescing) metrics to make sure you're efficiently utilizing the bytes transferred. Also consider whether it is possible to do more work per memory access (kernel fusion) or whether there are values you can (re)compute.

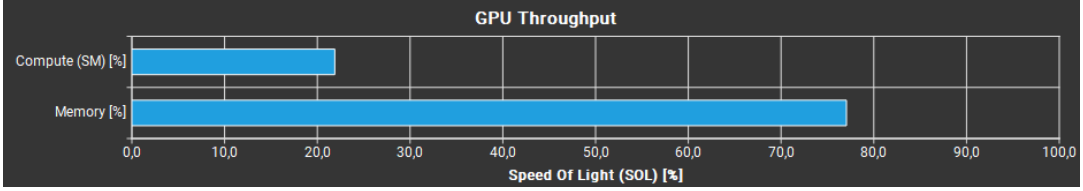

Figure 11

Theoretical Occupancy [%]	33,33	Block Limit Registers [block]	48
Theoretical Active Warps per SM [warp]	16	Block Limit Shared Mem [block]	16
Achieved Occupancy [%]	32,93	Block Limit Warps [block]	48
Achieved Active Warps Per SM [warp]	15,81	Block Limit SM [block]	16

Occupancy Limiters This kernel's theoretical occupancy (33.3%) is limited by the number of blocks that can fit on the SM. This kernel's theoretical occupancy (33.3%) is limited by the required amount of shared memory. See the CUDA Best Practices Guide for more details on optimizing occupancy.

Figure 12

After getting the profiling results, I followed the instructions and tried different block sizes. I noticed that when I increased the block size, memory throughput increased but occupany dropped and when I decreased the block size, memory througput dropped but occupancy increased.