Parallel Computing for GPUs using CUDA
Homework 4

Mustafa İzzet Muştu
504211564

June 2, 2023

Development and Test Environment

The operating system on the host machine is Ubuntu 22.04 and nvcc version is V12.1.105. Hardware information is as follows: AMD Ryzen 7 5800X, 32 GB 3400 MHz, MSI RTX 3060 12 GB.

1 Optimized Dense Matrix-Matrix Multiplication using GPUs

Algorithm 1 Tiled dense matrix-matrix multiplication kernel pseudocode

Require: $A, B, TILE\_WIDTH, Width$
Ensure: $C = AB$

```
shared_A[TILE_WIDTH][TILE_WIDTH]
shared_B[TILE_WIDTH][TILE_WIDTH]
bx ← blockIdx.x
by ← blockIdx.y
tx ← threadIdx.x
ty ← threadIdx.y
row ← ty + blockDim.y · by
column ← tx + blockDim.x · bx
result ← 0
for i = 0, i < (Width-1)/TILE_WIDTH + 1, i += 1 do
    shared_A[ty][tx] ← A[row · Width + i · TILE_WIDTH + tx]
    shared_B[ty][tx] ← B[(i · TILE_WIDTH + ty) · Width + column]
    __syncthreads()
    for j = 0, i < TILE_WIDTH, j += 1 do
        result+ = shared_A[ty][j] · shared_B[j][tx]
    end for
    __syncthreads()
end for
C[row · Width + column] ← result
```

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	0.0122	0.0006	0.0038	4.1031
2	0.0123	0.0006	0.0039	3.9985
3	0.0119	0.0005	0.0035	3.8982
4	0.0121	0.0006	0.0039	4.0941
5	0.0122	0.0006	0.0036	3.9992

Table 1: Results when the tile width is 32.

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

I implemented the tiling optimization only. The Pseudocode of the used kernel is shown in Algorithm 1. I set the block dimension which is also tile width 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 matrix 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 initialization times on the host and the GPU. Results can be seen in Table 1.

One thing to notice is that for the previous homework, I had the same configuration for the CPU multiplication function but I had better results (3.5 seconds). So, I realized that CPU performance is better with nvcc V12.1.105 on Windows 11. However, this situation may be unrelated to the OS, it may result better only because of further compiler optimizations. For the CUDA kernels, run times are very similar.

1.2 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. Compare your results against the very first version of your dense matrix-matrix multiplication implemented as part of homework 3.

I profiled the application using Nsight Compute. For the matrix multiplication kernel, I got 81.1% SM and memory throughput. For the occupancy, I achieved 66.75% occupancy. These results can be seen in Figure 1. 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 tile width determines the shared memory usage and block size, we can decrease tile width to 16.

After reducing the tile width, I achieved 98.47% occupancy and 95.35% throughput. These

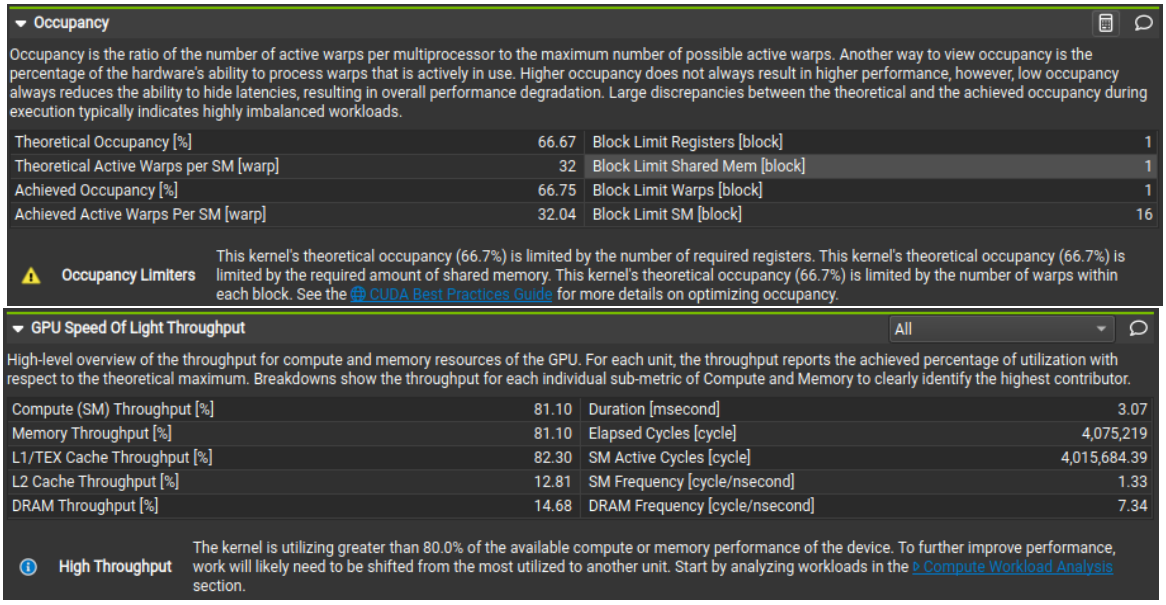


Figure 1: N=M=1024, tile width is 32.

results can be seen in Figure 2. For the run time, I did not notice any difference, so it is not included in the report.

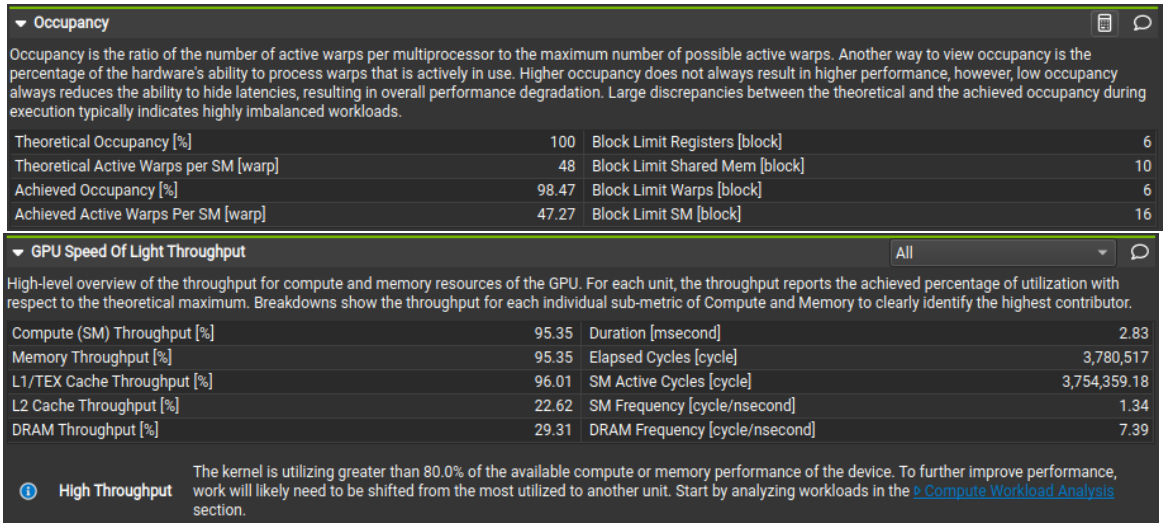


Figure 2: N=M=1024, tile width is 16.

1.3 Repeat the steps in (a) and (b) with matrix sizes of N=2048x2048, M=2048x2048 and N=4096x4096, M=4096x4096.

I set the tile width as 16 and did performance profiling. The throughput and the occupancy results can be seen in Figure 3 and Figure 4. For the run times, I run the kernel several times and measure the times. When N=M=2048, kernel time is approximately 33 seconds which is 2 seconds worse than the kernel time in homework 3. I discussed this situation in Section 1.1. Similarly, when N=M=4096, kernel time is approximately 286 seconds which is 30 seconds worse than the kernel time in homework 3.

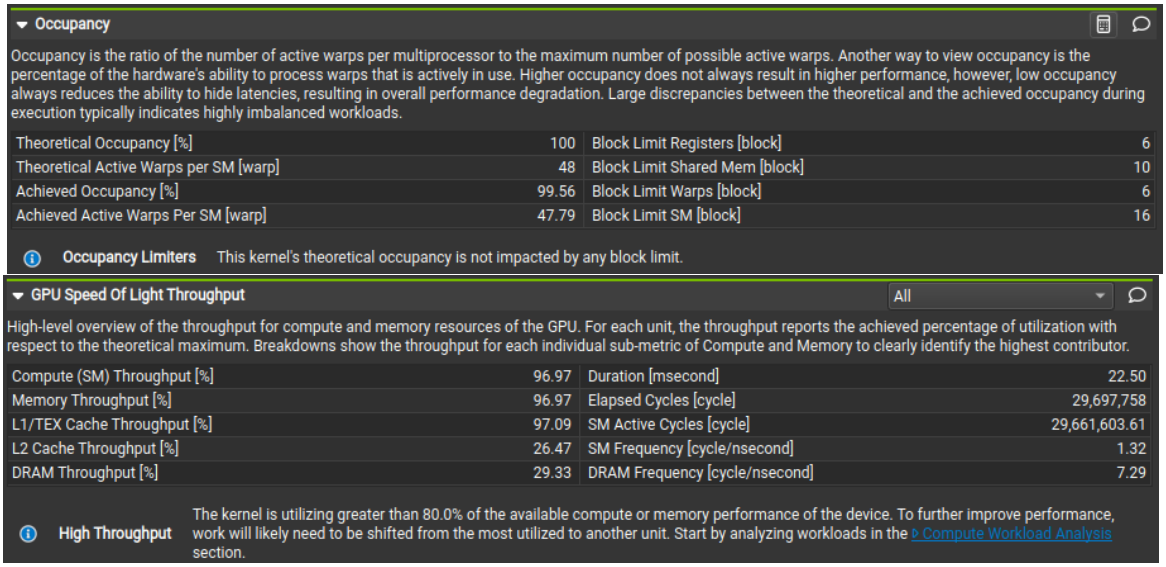


Figure 3: N=M=2048, tile width is 16.

Trial/Run Time (s)	Nonoptimized Kernel	Optimized Kernel
1	0.0279	0.0226
2	0.0283	0.0232
3	0.0284	0.0235
4	0.0276	0.0232
5	0.0282	0.0238

Table 2: Nonoptimized and optimized kernel run time comparison.

To compare these results with the homework 3 results, I set the tile width as 16, N=M=2048, and rerun both kernels on the same OS after compiling them with the same nvcc version. I did not any difference when it comes to the throughput and the occupancy. However, the run time for the optimized GPU kernel is slightly better than the nonoptimized kernel. The difference is shown in Table 2.

2 Sparse Matrix-Matrix Multiplication using GPUs

I utilized the cuSPARSE library for this part of the homework. I downloaded 494_bus, 685_bus, and 1138_bus matrices from the given link. These matrices contain 1666, 3249, and 4054 nonzero values respectively. For implementation, I read the second and the third arguments of the program as input matrix names. After, I read the input matrices by using these file names to the COO sparse matrix structure I created and then I convert these structures to the cuSPARSE COO format. To be able to multiply 2 sparse matrices, I also convert matrices from cuSPARSE COO to cuSPARSE CSR because only CSR matrix format is supported for multiplication.

2.1 Measure the running time of your kernel.

The running times for the CPU code and cuSPARSE library kernel are shown in Table 3. One thing to notice is that the GPU kernel run times do not change between N=M=494 and

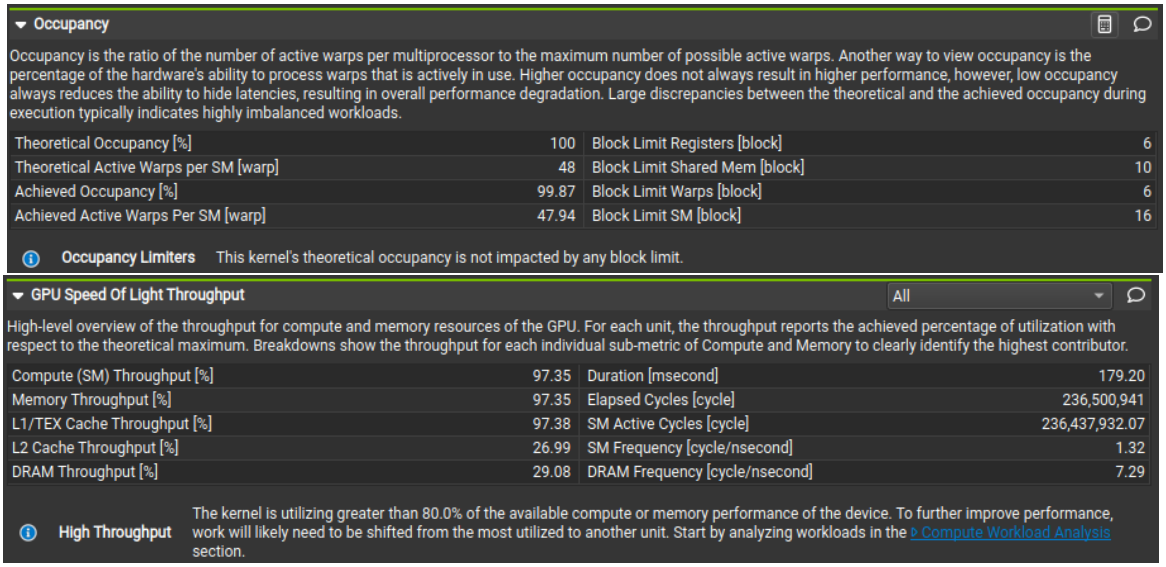


Figure 4: N=M=4096, tile width is 16.

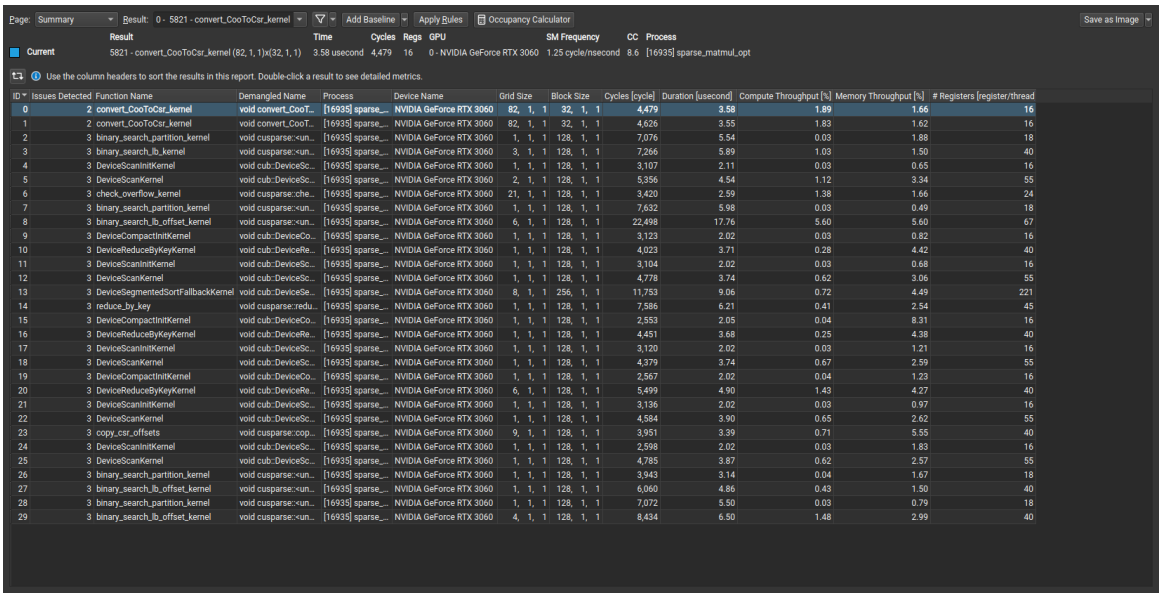
N,M/Run Time (s)	Sparse Matrix Multiplication on CPU	Sparse Matrix Multiplication on GPU
494	0.0363	0.0003
685	0.1575	0.0003
1138	0.2234	0.0003

Table 3: The run times for the CPU code and cuSPARSE library kernel.

N=M=1138 even though the latter contains almost as twice much as the former.

2.2 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.

Although kernel run times seem too low compared to the CPU implementation, the occupancy and throughput results are much worse. When I did performance profiling, I encountered 30 different kernels in Nsight Compute. Since the cuSPARSE library does not share the kernel names and does not let us change the block size, I could not continue the further optimizations. Kernels used in cuSPARSE library and information about them is shared in Figure 5.



ID	Issues Detected	Function Name	Demangled Name	Process	Device Name	Grid Size	Block Size	Cycles [cycle]	Duration [usecond]	Compute Throughput [%]	Memory Throughput [%]	# Registers [register/thread
0	2	convert_CooToCsr_kernel	void convert_CooT...	[16935] sparse_...	NVIDIA GeForce RTX 3060	82, 1, 1	32, 1, 1	4,479	3.58	1.89	1.66	16
1	2	convert_CooToCsr_kernel	void convert_CooT...	[16935] sparse_...	NVIDIA GeForce RTX 3060	82, 1, 1	32, 1, 1	4,626	3.55	1.83	1.62	16
2	3	binary_search_partition_kernel	void cusparse::un...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	7,076	5.54	0.03	1.88	18
3	3	binary_search_lb_kernel	void cusparse::un...	[16935] sparse_...	NVIDIA GeForce RTX 3060	3, 1, 1	128, 1, 1	7,266	5.89	1.03	1.50	40
4	3	DeviceScanInitKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	3,107	2.11	0.03	0.65	16
5	3	DeviceScanKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	2, 1, 1	128, 1, 1	5,356	4.54	1.12	3.34	55
6	3	check_overflow_kernel	void cusparse::che...	[16935] sparse_...	NVIDIA GeForce RTX 3060	21, 1, 1	128, 1, 1	3,420	2.59	1.38	1.66	24
7	3	binary_search_partition_kernel	void cusparse::un...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	7,632	5.98	0.03	0.49	18
8	3	binary_search_lb_offset_kernel	void cusparse::un...	[16935] sparse_...	NVIDIA GeForce RTX 3060	6, 1, 1	128, 1, 1	22,498	17.76	5.60	5.60	67
9	3	DeviceCompactInitKernel	void cub::DeviceCo...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	3,123	2.02	0.03	0.82	16
10	3	DeviceReduceByKeyKernel	void cub::DeviceRe...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	4,023	3.71	0.28	4.42	40
11	3	DeviceScanInitKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	3,104	2.02	0.03	0.68	16
12	3	DeviceScanKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	4,778	3.74	0.62	3.06	55
13	3	DeviceSegmentedSortFallbackKernel	void cub::DeviceSe...	[16935] sparse_...	NVIDIA GeForce RTX 3060	8, 1, 1	256, 1, 1	11,753	9.06	0.72	4.49	221
14	3	reduce_by_key	void cusparse::redu...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	7,586	6.21	0.41	2.54	45
15	3	DeviceCompactInitKernel	void cub::DeviceCo...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	2,553	2.05	0.04	8.31	16
16	3	DeviceReduceByKeyKernel	void cub::DeviceRe...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	4,451	3.68	0.25	4.38	40
17	3	DeviceScanInitKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	3,120	2.02	0.03	1.21	16
18	3	DeviceScanKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	4,379	3.74	0.67	2.59	55
19	3	DeviceCompactInitKernel	void cub::DeviceCo...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	2,567	2.02	0.04	1.23	16
20	3	DeviceReduceByKeyKernel	void cub::DeviceRe...	[16935] sparse_...	NVIDIA GeForce RTX 3060	6, 1, 1	128, 1, 1	5,499	4.90	1.43	4.27	40
21	3	DeviceScanInitKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	3,136	2.02	0.03	0.97	16
22	3	DeviceScanKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	4,584	3.90	0.65	2.62	55
23	3	copy_csr_offsets	void cusparse::cop...	[16935] sparse_...	NVIDIA GeForce RTX 3060	9, 1, 1	128, 1, 1	3,951	3.39	0.71	5.55	40
24	3	DeviceScanInitKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	2,598	2.02	0.03	1.83	16
25	3	DeviceScanKernel	void cub::DeviceSc...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	4,785	3.87	0.62	2.57	55
26	3	binary_search_partition_kernel	void cusparse::un...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	3,943	3.14	0.04	1.67	18
27	3	binary_search_lb_offset_kernel	void cusparse::un...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	6,060	4.86	0.43	1.50	40
28	3	binary_search_partition_kernel	void cusparse::un...	[16935] sparse_...	NVIDIA GeForce RTX 3060	1, 1, 1	128, 1, 1	7,072	5.50	0.03	0.79	18
29	3	binary_search_lb_offset_kernel	void cusparse::un...	[16935] sparse_...	NVIDIA GeForce RTX 3060	4, 1, 1	128, 1, 1	8,434	6.50	1.48	2.99	40

Figure 5: The cuSPARSE kernels and related information.