

## GPU programming - assignment 2

Dr. Cherif Salama

### Team members:

1. Mina Ashraf Gamil - 900182973
2. Mohamed Ayman Mohamed - 900182267

**GPU Model:** GeForce RTX 3090

```
Fri Nov  4 19:33:12 2022
+-----+
| NVIDIA-SMI 520.61.05      Driver Version: 520.61.05      CUDA Version: 11.8      |
+-----+
| GPU  Name      Persistence-M| Bus-Id      Disp.A  | Volatile Uncorr. ECC  | |
| Fan  Temp  Perf  Pwr:Usage/Cap|          Memory-Usage | GPU-Util  Compute M.  |
|                               |               |             | MIG M. |
+-----+
| 0  NVIDIA GeForce ...  On   | 00000000:02:00.0  On    |                N/A | |
| 0%   41C     P8    25W / 350W |            366MiB / 24576MiB |       6%     Default |
|                               |               |             | N/A |
+-----+
+-----+
| Processes:                               |
| GPU  GI  CI      PID   Type  Process name        GPU Memory  |
|           ID  ID                  |                 Usage  |
|-----|
| 0    N/A N/A      950    G    /usr/lib/xorg/Xorg      194MiB  |
| 0    N/A N/A     1459    G    /usr/bin/gnome-shell    38MiB  |
| 0    N/A N/A    224141    G    ...RendererForSitePerProcess  131MiB  |
+-----+
● cse-p07-g07f@csep07g07f:~/GPU_programming_with_CUDA$ nvidia-smi --query-gpu=name
--format=csv,noheader
NVIDIA GeForce RTX 3090
```

## GeForce and TITAN Products

GPU	Compute Capability
GeForce RTX 3090 Ti	8.6
GeForce RTX 3090	8.6

We are using the 8.6

For the full-table, check this link:

<https://docs.nvidia.com/cuda/cuda-c-programming-guide/#features-and-technical-specifications>

Note that the KB and K units used in the following table correspond to 1024 bytes (i.e., a KiB) and 1024 respectively.

Table 15. Technical Specifications per Compute Capability

Technical Specifications	3.5	3.7	5.0	5.2	5.3	6.0	6.1	6.2	7.0	7.2	7.5	8.0	8.6	8.7	8.9	9.0
Maximum number of resident grids per device ( <a href="#">Concurrent Kernel Execution</a> )	32		16	128	32	16	128	16	128	16	128	128	128	128	128	128
Maximum dimensionality of grid of thread blocks								3								
Maximum x-dimension of a grid of thread blocks									$2^{31-1}$							
Maximum y- or z-dimension of a grid of thread blocks									65535							
Maximum dimensionality of a thread block									3							
Maximum x- or y-dimension of a block									1024							
Maximum z-dimension of a block									64							
Maximum number of threads per block									1024							
Warp size									32							
Maximum number of resident blocks per SM	16							32				16	32	16	24	32
Maximum number of resident warps per SM							64				32	64	48	64		
Maximum number of resident threads per SM							2048				1024	2048	1536	2048		
Number of 32-bit registers per SM	64 K	128 K							64 K							
Maximum number of 32-bit registers per thread block		64 K			32 K		64 K		32 K				64 K			
Maximum number of 32-bit registers per thread										255						
Maximum amount of shared memory per SM	48 KB	112 KB	64 KB	96 KB	64 KB	96 KB	64 KB	96 KB	64 KB	164 KB	100 KB	164 KB	100 KB	228 KB		
Maximum amount of shared memory per thread block <a href="#">34</a>				48 KB					96 KB	96 KB	64 KB	163 KB	99 KB	163 KB	99 KB	227 KB
Number of shared memory banks									32							
Maximum amount of local memory per thread									512 KB							
Constant memory size									64 KB							
Cache working set per SM for constant memory		8 KB			4 KB				8 KB							
Cache working set per SM for texture memory		Between 12 KB and 48 KB			Between 24 KB and 48 KB		32 - 128 KB		32 or 64 KB	28KB - 192 KB	28KB - 128 KB	28KB - 192 KB	28KB - 128 KB	28KB - 256 KB		
Maximum width for a 1D texture reference bound to a CUDA array		65536								131072						
Maximum width for a 1D texture reference bound to linear memory		$2^{27}$				$2^{28}$		$2^{27}$	$2^{28}$	$2^{27}$				$2^{28}$		
Maximum width and number of layers for a 1D layered texture reference		16384 x 2048								32768 x 2048						
Maximum width and height for a 2D texture reference bound to a CUDA array		65536 x 65536								131072 x 65536						

All the tests reported are for these matrices:

Dimensions of A: 1523x3232

Dimensions of B: 3232x1711

**Notes:**

The FLOPS calculation is a very complicated process, using counters is an option, but given the nature of the matrix-matrix multiplication problem, there is a theoretical proof for the number of calculations needed that neglects the simple loop variables.

**1 Answer**

Sorted by: Highest score (default) ▾

▲ There are  $n \times m$  elements in the output matrix. Each of them is obtained by  $p$  multiplications (1 element from the first matrix and 1 from the second), then summing up. Since you have  $p$  products, you add  $p - 1$  of them to the first one. So the number of operations for one element in the output matrix is  $p$  multiplications and  $p - 1$  additions, meaning  $2p - 1$  FLOPS. Then for all elements, you get  $n \times m \times (2p - 1)$  FLOPS.

4

[Share](#) [Cite](#) [Edit](#) [Follow](#)

answered Jan 17, 2020 at 22:51



Andrei

34.1k 5 24 48

**A more complete formulations of the FLOPS problem for the common matrices and vector problems is here:**

**vector-vector operations** ( $x, y \in \mathbf{R}^n$ )

- inner product  $x^T y$ :  $2n - 1$  flops (or  $2n$  if  $n$  is large)
- sum  $x + y$ , scalar multiplication  $\alpha x$ :  $n$  flops

**matrix-vector product**  $y = Ax$  with  $A \in \mathbf{R}^{m \times n}$

- $m(2n - 1)$  flops (or  $2mn$  if  $n$  large)
- $2N$  if  $A$  is sparse with  $N$  nonzero elements
- $2p(n + m)$  if  $A$  is given as  $A = UV^T$ ,  $U \in \mathbf{R}^{m \times p}$ ,  $V \in \mathbf{R}^{n \times p}$

**matrix-matrix product**  $C = AB$  with  $A \in \mathbf{R}^{m \times n}$ ,  $B \in \mathbf{R}^{n \times p}$

- $mp(2n - 1)$  flops (or  $2mnp$  if  $n$  large)
- less if  $A$  and/or  $B$  are sparse
- $(1/2)m(m + 1)(2n - 1) \approx m^2n$  if  $m = p$  and  $C$  symmetric

# Task 1

We tested it with blocks of 16x16 threads, and 32x32

We also computed the GFLOPS for both the GPU and the CPU versions as well as the total time with and without the data transfer times.

## Run 1:

Number of floating points operations needed for this matrix: **1.68416e+10**

CPU took: **38.4671 seconds**

Thus, CPU GFLOPS: **4.37819e+08**

GPU with blockSize: **16x16**

Total number of blocks: **10272**

Total number of threads: **2629632**

GPU matrix multiplication took **209730080** nanoseconds (**WITHOUT DATA TRANSFER**)

GPU matrix multiplication took **347687000** nanoseconds (**WITH DATA TRANSFER**)

Therefore, data transfer takes about:  $(347687000 - 209730080) / 347687000 = 0.39678$  or **40% of the whole operation!**

**GPU GFLOPS without data transfer: 4.84391e+11**

This is (**1106.37** times the GFLOPS of CPU!)

**GPU GFLOPS with data transfer: 8.03014e+10 (still 183.41 times the CPU)**

```

● ➔ GPU ./gpu_assignment

Starting...
Dimensions of A: 1523x3232
Dimensions of B: 3232x1711
Starting CPU matrix multiplication...
CPU matrix multiplication took 38467801451 nanoseconds

FLOPS: : 1.68416e+10

Time in seconds: 38.4671

CPU GFLOPS: 4.37819e+08

Starting GPU matrix multiplication...
Total number of blocks: 10272
Total number of threads: 2629632

GPU matrix multiplication took 209730080 nanoseconds (WITHOUT DATA TRANSFER)
GPU matrix multiplication took 347687000 nanoseconds (WITH DATA TRANSFER)

FLOPS: : 1.68416e+10

Time in seconds: 0.20973008

GPU GFLOPS of kernel only: 4.84391e+11

FLOPS: : 1.68416e+10

Time in seconds: 0.347687

GPU GFLOPS with copy: 8.03014e+10

Comparing matrices...

Matrices are equal
○ ➔ GPU □

```

## GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability)

Page: Details Result: 0 - 115 - MatrixMulKernel Add Baseline Apply Rules Occupancy Calculator Save as PDF

Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	115 - MatrixMulKernel (107, 96, 1)x(16, 16, 1)	38.00 msecound	0r'91Y'YEA	6	0 - NVIDIA GeForce RTX 3090	1.39 cycle/nsecound	8.6 [252187] task1

**GPU Speed Of Light Throughput**

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.

Compute (SM) Throughput [%]	Duration [msecond]	Memory Throughput [%]	Elapsed Cycles [cycle]	L1/TEX Cache Throughput [%]	SM Active Cycles [cycle]	L2 Cache Throughput [%]	SM Frequency [cycle/nsecound]	DRAM Throughput [%]	DRAM Frequency [cycle/nsecound]
97.09	38.00	52,962,348	52,739,091.17	25.12	0	13.34	1.39	12.74	9.48

**High Throughput** The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the [Compute Workload Analysis](#) section.

**Launch Statistics**

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	Function Cache Configuration	cudaFuncCachePreferNone
Registers Per Thread [register/thread]	40 Static Shared Memory Per Block [byte/block]	0
Block Size	256 Dynamic Shared Memory Per Block [byte/block]	0
Threads [thread]	2,629,632 Driver Shared Memory Per Block [Kbyte/block]	1.02
Waves Per SM	20.88 Shared Memory Configuration Size [Kbyte]	8.19

**Occupancy**

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	Block Limit Registers [block]	6
Theoretical Active Warps per SM [warp]	48 Block Limit Shared Mem [block]	8
Achieved Occupancy [%]	98.10 Block Limit Warps [block]	6
Achieved Active Warps Per SM [warp]	47.09 Block Limit SM [block]	16

**Occupancy Limiters** This kernel's theoretical occupancy is not impacted by any block limit.

You can see a few key notes from the NSight-compute analysis:  
Our theoretical occupancy was: 100% so our kernel is not impacted by any block limits  
The achieved occupancy was 98.10% which is really good.

Caching was utilized although our task implementation was pure brute-force, which made it a little faster, but still nothing compared to the second and third tasks.

### **Run 2: (32x32 blocks)**

Number of floating points operations needed for this matrix: **1.68416e+10**

CPU took: **38.8542 seconds**

**Thus, CPU GFLOPS: 4.33457e+08 (slightly less than last run, which is normal)**

GPU with blockSize: **32x32**

Total number of blocks: **2592**

Total number of threads: **2654208**

GPU matrix multiplication took **126222238** nanoseconds (**WITHOUT DATA TRANSFER**)

GPU matrix multiplication took **314460000** nanoseconds (**WITH DATA TRANSFER**)

Therefore, data transfer takes about: **(314460000-126222238 / 314460000) = 0.5986 or**

**60% of the whole process**

**GPU GFLOPS without data transfer: 5.35574e+11**

This is **(1235.58 times the GFLOPS of CPU!)**

**GPU GFLOPS with data transfer: 1.33428e+11 (still 307.8 times the CPU)**

**Therefore:**

**The 32x32 blocks is faster than the 16x16 block size by 10.57%**

**5.35574e+11 / 4.84391e+11 = 1.10566463869**

**GPU Utilization profiling and occupancy analysis: (ZOOM IN for more readability)**

Page: Details Result: 0 - 115 - MatrixMulKernel

Result	Time	Cycles	Regs	GPU	SM Frequency	CC Process
Current	115 - MatrixMulKernel (54, 48, 1)x(32, 32, 1)	38.09 msecound	0x11111111	0 - NVIDIA GeForce RTX 3090	1.39 cycle/nsecond	8.6 [250832] task1

**GPU Speed Of Light Throughput**

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.

Compute (SM) Throughput [%]	97.61 Duration [msecond]	38.09
Memory Throughput [%]	25.01 Elapsed Cycles [cycle]	53,133,919
L1/TEX Cache Throughput [%]	25.19 SM Active Cycles [cycle]	52,762,941.48
L2 Cache Throughput [%]	10.33 SM Frequency [cycle/nsecond]	1.39
DRAM Throughput [%]	5.81 DRAM Frequency [cycle/nsecond]	9.49

**High Throughput** The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the [Compute Workload Analysis](#) section.

**Launch Statistics**

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	2,592 Function Cache Configuration	cudaFuncCachePreferNone
Registers Per Thread [register/thread]	40 Static Shared Memory Per Block [byte/block]	0
Block Size	1,024 Dynamic Shared Memory Per Block [byte/block]	0
Threads [thread]	2,654,208 Driver Shared Memory Per Block [kbyte/block]	1.02
Waves Per SM	31.61 Shared Memory Configuration Size [kbyte]	8.19

**Occupancy**

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	66.67 Block Limit Registers [block]	1
Theoretical Active Warps per SM [warp]	32 Block Limit Shared Mem [block]	8
Achieved Occupancy [%]	65.43 Block Limit Warps [block]	1
Achieved Active Warps Per SM [warp]	31.41 Block Limit SM [block]	16

**Occupancy Limiters** This kernel's theoretical occupancy (66.7%) is limited by the number of required registers. This kernel's theoretical occupancy (66.7%) is limited by the number of warps within each block. See the [CUDA Best Practices Guide](#) for more details on optimizing occupancy.

You can see a few key notes from the NSight-compute analysis:

Our theoretical occupancy was: 66.67%, which is very low compared to the 16x16

The achieved occupancy was even lower at 65.43%.

It turns out that we are heavily impacted by the number of registers required per block. So, we tried a better implementation that doesn't pass the dimensions of the A to function parameters, but uses the global #defines instead to see the difference.

Yet, it remains faster because occupancy doesn't mean higher performance, but it might be more suitable for hiding latencies.

```
● ➔ GPU ./gpu_assignment

Starting...
Dimensions of A: 1523x3232
Dimensions of B: 3232x1711
Starting CPU matrix multiplication...
CPU matrix multiplication took 3.8541e+10 nanoseconds

FLOPS: : 1.68416e+10

Time in seconds: 38.8542

CPU GFLOPS: 4.37819e+08

Starting GPU matrix multiplication...
Total number of blocks: 2592
Total number of threads: 2654208

GPU matrix multiplication took 126222238 nanoseconds (WITHOUT DATA TRANSFER)
GPU matrix multiplication took 314460000 nanoseconds (WITH DATA TRANSFER)

FLOPS: : 1.68416e+10

Time in seconds: 0.12622

GPU GFLOPS of kernel only: 5.35574e+11

FLOPS: : 1.68416e+10

Time in seconds: 0.314460000

GPU GFLOPS with copy: 1.33428e+11

Comparing matrices...

Matrices are equal
```

# Task 2

## RUN 1 (16x16 blocks):

This time, we implemented the matrix-matrix multiplication code using tiling and shared memory to see how much speedup would we get from reducing global memory accesses.

Dimensions of A: **1523x3232**

Dimensions of B: **3232x1711**

CPU matrix multiplication took 37857235971 nanoseconds (37.8572 seconds)

FLOPS: 1.68416e+10 (**from the equation**)

CPU GFLOPS: **4.44872e+08**

For GPU code with 16x16 blocks:

Total number of blocks: **10272**

Total number of threads: **2629632**

GPU matrix multiplication took **190998933** nanoseconds or **0.190999** seconds

FLOPS: **1.68416e+10**

Time in seconds for the kernel alone without data transfer: **0.0687013**

GPU GFLOPS of kernel only without data transfer: **2.45143e+11**

GPU GFLOPS with copy: **8.81766e+10**

**Speedup over CPU: 551 and 198 for kernel only and kernel with data transfer respectively**

## GPU utilization and occupancy analysis:

Page: Details Result: 0 - 115 - MatrixMulKernel Add Baseline Apply Rules Occupancy Calculator Save as PDF

Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current	115 - MatrixMulKernel (107, 96, 1)x(16, 16, 1)	38.55 msecound	0x7FFF00000000	0 - NVIDIA GeForce RTX 3090	1.39 cycle/nsecound	8.6 [251520] task2	

► GPU Speed Of Light Throughput

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.

Compute (SM) Throughput [%]	Memory Throughput [%]	L1/TEX Cache Throughput [%]	L2 Cache Throughput [%]	DRAM Throughput [%]	
96.37 Duration [msecound]	27.83 Elapsed Cycles [cycle]	28.00 SM Active Cycles [cycle]	11.50 SM Frequency [cycle/nsecound]	12.46 DRAM Frequency [cycle/nsecound]	38.55 53,778,477 53,466,385.41 1.39 9.49

**High Throughput** The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the [Compute Workload Analysis](#) section.

► Launch Statistics

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	Registers Per Thread [register/thread]	Block Size	Threads [thread]	Waves Per SM	Function Cache Configuration	cudaFuncCachePreferNone
10,272	35	256	2,629,632	20.88	Static Shared Memory Per Block [Kbyte/block] Dynamic Shared Memory Per Block [byte/block] Driver Shared Memory Per Block [Kbyte/block] Shared Memory Configuration Size [Kbyte]	4.10 0 1.02 65.54

► Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	Theoretical Active Warps per SM [warp]	Achieved Occupancy [%]	Achieved Active Warps Per SM [warp]	Block Limit Registers [block]	Block Limit Shared Mem [block]	Block Limit Warps [block]	Block Limit SM [block]
100	48	98.65	47.35	100	48	100	6
							12
							6
							16

**Occupancy Limiters** This kernel's theoretical occupancy is not impacted by any block limit.

You can see we have a theoretical occupancy of 100% and achieved 98.65%

```
● → GPU ./gpu_assignment

Starting...
Dimensions of A: 1523x3232
Dimensions of B: 3232x1711
Starting CPU matrix multiplication...
CPU matrix multiplication took 37857235971 nanoseconds

FLOPS: : 1.68416e+10

Time in seconds: 37.8572

CPU GFLOPS: 4.44872e+08

Starting GPU matrix multiplication...
Total number of blocks: 10272
Total number of threads: 2629632

GPU matrix multiplication took 190998933 nanoseconds or 0.190999 seconds

FLOPS: 1.68416e+10

Time in seconds for the kernel alone without data transfer: 0.0687013

GPU GFLOPS of kernel only without data transfer: 2.45143e+11

FLOPS: 1.68416e+10
GPU GFLOPS with copy: 8.81766e+10

Comparing matrices...

Matrices are equal
○ → GPU
```

## RUN 2 (32x32 blocks):

Dimensions of A: 1523x3232

Dimensions of B: 3232x1711

Starting CPU matrix multiplication...

CPU matrix multiplication took 39147475633 nanoseconds (39.1475 seconds)

**FLOPS:** 1.68416e+10    **CPU GFLOPS:** 4.3021e+08

## For GPU (32x32)

Total number of blocks: 2592

Total number of threads: 2654208

GPU matrix multiplication took **134484166** nanoseconds

FLOPS: 1.68416e+10

Time in seconds: **0.0343415**

GPU GFLOPS of kernel only: **4.90416e+11**

Time in seconds with data transfer: **0.134484**

GPU GFLOPS with copy: **1.25231e+11**

**Speedup over CPU: 1139.9 times without the transfer and 291 with the transfer**

**Speedup over the 16x16 is: 200% (2x)**

## Utilization analysis:

Page: Details Result: 0 - 115 - MatrixMulKernel Add Baseline Apply Rules Occupancy Calculator Save as PDF

Result	Time	Cycles	Regs	GPU	SM Frequency	CC	Process
Current 115 - MatrixMulKernel (54, 48, 1)x(32, 32, 1)	41.49 msecound	0Y'XP·'T11	70	0 - NVIDIA GeForce RTX 3090	1.39 cycle/nsecond	8.6	[251075] task2

► GPU Speed Of Light Throughput All

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.

Compute (SM) Throughput [%]	Duration [msecound]	41.49
Memory Throughput [%]	Elapsed Cycles [cycle]	57,820,361
L1/TEX Cache Throughput [%]	SM Active Cycles [cycle]	57,109,038.50
L2 Cache Throughput [%]	SM Frequency [cycle/nsecond]	1.39
DRAM Throughput [%]	DRAM Frequency [cycle/nsecond]	9.49

High Throughput The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the [Compute Workload Analysis](#) section.

► Launch Statistics

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	Function Cache Configuration	cudaFuncCachePreferNone
Registers Per Thread [register/thread]	35 Static Shared Memory Per Block [Kbyte/block]	16.38
Block Size	1,024 Dynamic Shared Memory Per Block [byte/block]	0
Threads [thread]	2,654,208 Driver Shared Memory Per Block [Kbyte/block]	1.02
Waves Per SM	31.61 Shared Memory Configuration Size [Kbyte]	32.77

► Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	Block Limit Registers [block]	1
Theoretical Active Warps per SM [warp]	32 Block Limit Shared Mem [block]	1
Achieved Occupancy [%]	66.65 Block Limit Warps [block]	1
Achieved Active Warps Per SM [warp]	31.99 Block Limit SM [block]	16

Occupancy Limiters This kernel's theoretical occupancy (66.7%) is limited by the number of required registers. This kernel's theoretical occupancy (66.7%) is limited by the required amount of shared memory. This kernel's theoretical occupancy (66.7%) is limited by the number of warps within each block. See the [CUDA Best Practices Guide](#) for more details on optimizing occupancy.

We now have theoretical occupancy of 66.67 and we are being limited by the size of the shared memory. This explains why the first version of the code might be still competitive at this size. You can notice the throughput is down to 90.50% due to this overhead.

# Task 3

Run 1:

**Block size 16x16 and granularity is 2**

Dimensions of A: 1523x3232

Dimensions of B: 3232x1711

CPU matrix multiplication took 37942277021 nanoseconds (**37.9423** seconds)

FLOPS: **1.68416e+10**

CPU GFLOPS: **4.43875e+08**

Total number of blocks: **5184**

Total number of threads: **1327104**

GPU matrix multiplication took 186703162 nanoseconds (0.1867 seconds with data transfer)

GPU GFLOPS with copy: 9.02054e+10

Time in seconds: **0.035147** for the kernel only

GPU GFLOPS of kernel only: **4.79177e+11**

GPU speedups is: **1079.531 without data transfer over the CPU**

```
Matrices are equal
● ➔ GPU ./gpu_assignment

Starting...
Dimensions of A: 1523x3232
Dimensions of B: 3232x1711
Starting CPU matrix multiplication...
CPU matrix multiplication took 37942277021 nanoseconds

FLOPS: : 1.68416e+10

Time in seconds: 37.9423

CPU GFLOPS: 4.43875e+08

Starting GPU matrix multiplication...
Total number of blocks: 5184
Total number of threads: 1327104

GPU matrix multiplication took 186703162 nanoseconds (0.1867 seconds with data transfer)

FLOPS: : 1.68416e+10
GPU GFLOPS with copy: 9.02054e+10

Time in seconds: 0.035147 for the kernel only

FLOPS: : 1.68416e+10
GPU GFLOPS of kernel only: 4.79177e+11

Comparing matrices...

Matrices are equal
```

CPU █

## Utilization and occupancy analysis:

Another 100% theoretical occupancy and around 97% achieved.

You can also see it has very high compute SM throughput. **It is worth noting that it is still the slowest implementation of all the tasks!**

Page: Details Result: 0 - 115 - MatrixMulKernel Add Baseline Apply Rules Occupancy Calculator Save as PDF

Result Time Cycles Regs GPU SM Frequency CC Process

Current 115 - MatrixMulKernel (54,96,1)x(16,16,1) 38.59 msecound 0x1A1C1111 0 - NVIDIA GeForce RTX 3090 1.39 cycle/nsecond 8.6 [251963] task3

GPU Speed Of Light Throughput All

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.

Compute (SM) Throughput [%]	97.18 Duration [msecond]	38.59
Memory Throughput [%]	18.02 Elapsed Cycles [cycle]	53,824,711
L1/TEX Cache Throughput [%]	18.25 SM Active Cycles [cycle]	53,161,016.88
L2 Cache Throughput [%]	8.45 SM Frequency [cycle/nsecond]	1.39
DRAM Throughput [%]	8.00 DRAM Frequency [cycle/nsecond]	9.49

High Throughput The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the [Compute Workload Analysis](#) section.

Launch Statistics

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	5,184 Function Cache Configuration	cudaFuncCacheNone
Registers Per Thread [register/thread]	40 Static Shared Memory Per Block [Kbyte/block]	6.14
Block Size	256 Dynamic Shared Memory Per Block [byte/block]	0
Threads [thread]	1,327,104 Driver Shared Memory Per Block [Kbyte/block]	1.02
Waves Per SM	10.54 Shared Memory Configuration Size [Kbyte]	65.54

Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	100 Block Limit Registers [block]	6
Theoretical Active Warps per SM [warp]	48 Block Limit Shared Mem [block]	9
Achieved Occupancy [%]	96.93 Block Limit Warps [block]	6
Achieved Active Warps Per SM [warp]	46.53 Block Limit SM [block]	16

Occupancy Limiters This kernel's theoretical occupancy is not impacted by any block limit.

## Run 2:

Block size 16x16 and granularity is 4

Dimensions of A: 1523x3232

Dimensions of B: 3232x1711

CPU matrix multiplication took 37351129958 nanoseconds (37.3511 seconds)

**FLOPS: 1.68416e+10**

Time in seconds: CPU GFLOPS: **4.509e+08**

**GPU 16x16 blocks and granularity is 4**

Total number of blocks: **2592**

Total number of threads: **663552**

GPU matrix multiplication took **158775880** nanoseconds (0.158776 seconds)

Time in seconds without data transfer: **0.031878**

GPU GFLOPS of kernel only: **5.28315e+11 (FASTEAST ONE SO FAR)**

GPU GFLOPS with copy: **1.06072e+11**

**Speedup over CPU: 1190.23 without data transfer and 238.96 with data transfer**

Utilization and occupancy analysis:

100% theoretical occupancy and 97.43% compute throughput and 16.17% memory throughput

Page: Details Result: 0 - 115 - MatrixMulKernel Add Baseline Apply Rules Occupancy Calculator Save as PDF

Result Time Cycles Regs GPU SM Frequency CC Process

Current 115 - MatrixMulKernel (27, 96, 1)x(16, 16, 1) 38.49 msecnd 0 0 - NVIDIA GeForce RTX 3090 1.39 cycle/nsecnd 8.6 [250632] task3

GPU Speed Of Light Throughput All

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.

Compute (SM) Throughput [%]	Duration [msecnd]	38.49
Memory Throughput [%]	Elapsed Cycles [cycle]	53,693,726
L1/TEX Cache Throughput [%]	SM Active Cycles [cycle]	53,027,999.11
L2 Cache Throughput [%]	SM Frequency [cycle/nsecnd]	1.39
DRAM Throughput [%]	DRAM Frequency [cycle/nsecnd]	9.49

High Throughput The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the [Compute Workload Analysis](#) section.

Launch Statistics

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	Function Cache Configuration	cudaFuncCachePreferNone
Registers Per Thread [register/thread]	40 Static Shared Memory Per Block [kbyte/block]	10.24
Block Size	256 Dynamic Shared Memory Per Block [byte/block]	0
Threads [thread]	663,552 Driver Shared Memory Per Block [kbyte/block]	1.02
Waves Per SM	5.27 Shared Memory Configuration Size [kbyte]	102.40

Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	Block Limit Registers [block]	6
Theoretical Active Warps per SM [warp]	48 Block Limit Shared Mem [block]	9
Achieved Occupancy [%]	93.49 Block Limit Warps [block]	6
Achieved Active Warps Per SM [warp]	44.88 Block Limit SM [block]	16

Occupancy Limiters This kernel's theoretical occupancy is not impacted by any block limit.

```
● ➔ GPU ./gpu_assignment

Starting...
Dimensions of A: 1523x3232
Dimensions of B: 3232x1711
Starting CPU matrix multiplication...
CPU matrix multiplication took 37351129958 nanoseconds

FLOPS: : 1.68416e+10

Time in seconds: 37.3511

CPU GFLOPS: 4.509e+08

Starting GPU matrix multiplication...
Total number of blocks: 2592
Total number of threads: 663552

GPU matrix multiplication took 158775880 nanoseconds (0.158776 seconds with data transfer)

FLOPS: : 1.68416e+10
GPU GFLOPS with copy: 1.06072e+11

Time in seconds: 0.031878 for the kernel only

FLOPS: : 1.68416e+10
GPU GFLOPS of kernel only: 5.28315e+11

Comparing matrices...

Matrices are equal
```

### Run 3:

#### **Block size 32x32 and granularity is 2**

Dimensions of A: 1523x3232

Dimensions of B: 3232x1711

CPU matrix multiplication took 38415860821 nanoseconds (38.4159 seconds)

**FLOPS: 1.68416e+10**

CPU GFLOPS: **4.38403e+08**

Total number of blocks: 1296

Total number of threads: 1327104

GPU matrix multiplication took 144913071 nanoseconds (0.144913 seconds)

Time in seconds without data transfer: **0.0359325**

GPU GFLOPS of kernel only: **4.68702e+11 (much slower than 16x16, profiler will show why)**

GPU GFLOPS with copy: **1.16219e+11**

Speedup over CPU: **1055.93** and **261.82** with and without data transfer

## Utilization and occupancy analysis:

Our kernel has 66.67% occupancy and is being limited by the number of registers this time. Otherwise, everything is looking good with high SM throughput and same as usual memory throughput.

Page: Details Result: 0 - 115 - MatrixMulKernel Add Baseline Apply Rules Occupancy Calculator Save as PDF

Result Time Cycles Regs GPU SM Frequency CC Process

Current 115 - MatrixMulKernel (27, 48, 1)x(32, 32, 1) 39.26 msecound 0x'Y1A'11E 0 - NVIDIA GeForce RTX 3090 1.39 cycle/nsecond 8.6 [251329] task3

**GPU Speed Of Light Throughput**

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.

Compute (SM) Throughput [%]	95.63	Duration [msecond]	39.26
Memory Throughput [%]	16.31	Elapsed Cycles [cycle]	54,718,114
L1/TEX Cache Throughput [%]	16.51	SM Active Cycles [cycle]	54,033,552.09
L2 Cache Throughput [%]	4.08	SM Frequency [cycle/nsecond]	1.39
DRAM Throughput [%]	2.30	DRAM Frequency [cycle/nsecond]	9.49

**High Throughput** The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the [Compute Workload Analysis](#) section.

**Launch Statistics**

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

Grid Size	1,296	Function Cache Configuration	cudaFuncCacheNone
Registers Per Thread [register/thread]	40	Static Shared Memory Per Block [Kbyte/block]	24.58
Block Size	1,024	Dynamic Shared Memory Per Block [byte/block]	0
Threads [thread]	1,327,104	Driver Shared Memory Per Block [Kbyte/block]	1.02
Waves Per SM	15.80	Shared Memory Configuration Size [Kbyte]	32.77

**Occupancy**

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

Theoretical Occupancy [%]	66.67	Block Limit Registers [block]	1
Theoretical Active Warps per SM [warp]	32	Block Limit Shared Mem [block]	1
Achieved Occupancy [%]	66.65	Block Limit Warps [block]	1
Achieved Active Warps Per SM [warp]	31.99	Block Limit SM [block]	16

**Occupancy Limiters** This kernel's theoretical occupancy (66.7%) is limited by the number of required registers. This kernel's theoretical occupancy (66.7%) is limited by the required amount of shared memory. This kernel's theoretical occupancy (66.7%) is limited by the number of warps within each block. See the [CUDA Best Practices Guide](#) for more details on optimizing occupancy.

```
● → GPU ./gpu_assignment

Starting...
Dimensions of A: 1523x3232
Dimensions of B: 3232x1711
Starting CPU matrix multiplication...
CPU matrix multiplication took 38415860821 nanoseconds

FLOPS: : 1.68416e+10

Time in seconds: 38.4159

CPU GFLOPS: 4.38403e+08

Starting GPU matrix multiplication...
Total number of blocks: 1296
Total number of threads: 1327104

GPU matrix multiplication took 144913071 nanoseconds (0.144913 seconds with data transfer)

FLOPS: : 1.68416e+10
GPU GFLOPS with copy: 1.16219e+11

Time in seconds: 0.0359325 for the kernel only

FLOPS: : 1.68416e+10
GPU GFLOPS of kernel only: 4.68702e+11

Comparing matrices...

Matrices are equal
```

Run 4:

### Block size 32x32 and granularity is 4

Dimensions of A: 1523x3232

Dimensions of B: 3232x1711

CPU matrix multiplication took 39488929569 nanoseconds ( 39.4889 seconds)

**FLOPS: 1.68416e+10**

CPU GFLOPS: **4.2649e+08**

Starting GPU matrix multiplication...

Total number of blocks: 672

Total number of threads: 688128

GPU matrix multiplication took 171116502 nanoseconds (0.171116 seconds)

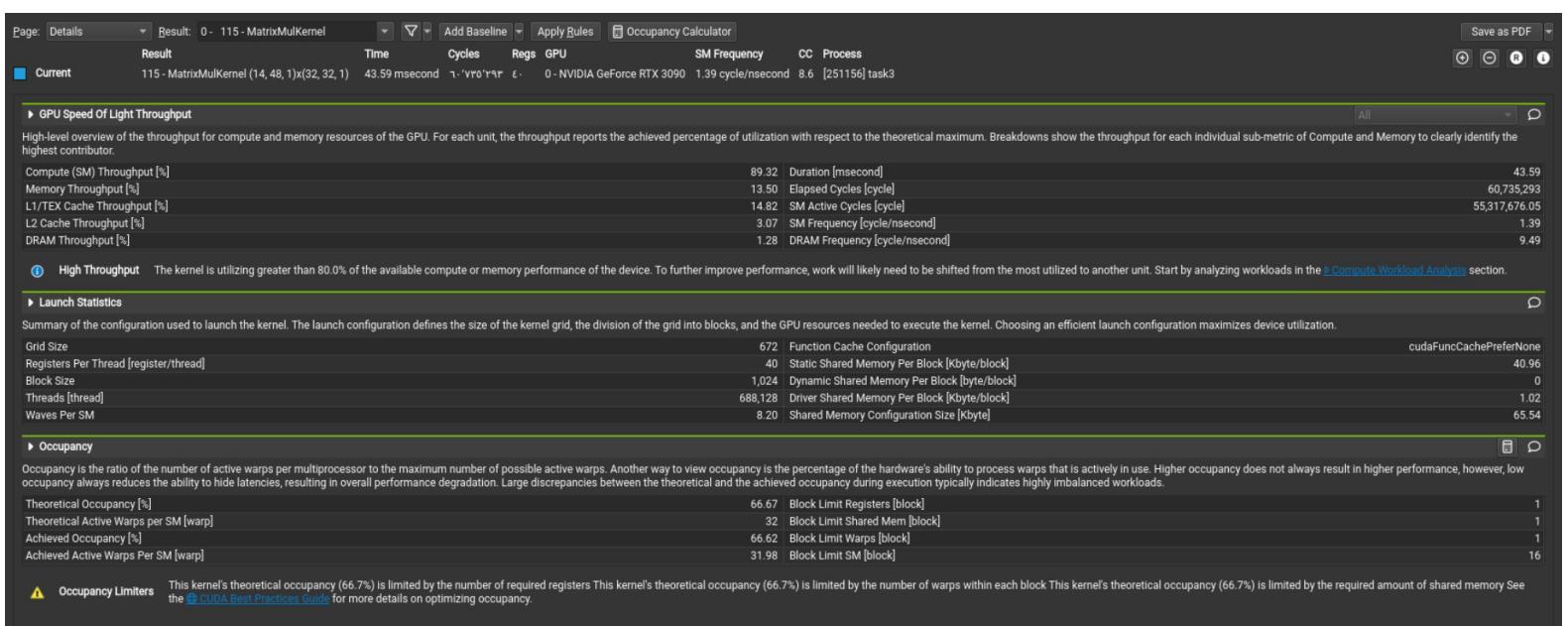
Time in seconds of kernel only: **0.0359663**

GPU GFLOPS of kernel only: **4.68262e+11**

GPU GFLOPS with copy: **9.8422e+10**

Speedup of GPU over CPU is: **1054.94** and **221.73**

Utilization and occupancy analysis:



Occupancy is limited due to the number of registers again and we have a 89.32 compute SM throughput.

## **More tests:**

**Problem:** We noticed that sometimes, task 1, the simplest version without any shared memory or granularity, can outperform the more optimized versions, despite making sure both are implemented 100% correctly.

So, to verify this, we need to put both versions under bigger stress and larger tests so that the small differences would be magnified and see if the optimizations actually make it better or not.

So, we tested both with these matrices:

**Matrix A: 2532 x 3232**

**Matrix B: 3232 x 4711**

**Task 1:** 16x16 block size

CPU matrix multiplication took 193581901451 nanoseconds (193.582 seconds)

FLOPS: **7.68183e+10**

CPU GFLOPS: **3.96826e+08**

Total number of blocks: 46610

Total number of threads: 11932160

GPU matrix multiplication took 299241542 ns (0.299s with copy and allocation)

Time in seconds: 0.150121 for kernel only

**GPU GFLOPS of kernel only: 5.11711e+11**

**GPU GFLOPS with copy: 2.5671e+11**

Speedups over CPU: **1289.509 without data transfer**

**And 646.90 with data transfer (This makes it the fastest one test so far)**

```
● ➔ GPU ./gpu_assignment
Starting...
Dimensions of A: 2523x3232
Dimensions of B: 3232x4711

Starting CPU matrix multiplication...
CPU matrix multiplication took 193581901451 nanoseconds

FLOPS: 7.68183e+10

Time in seconds: 193.582

CPU GFLOPS: 3.96826e+08

Starting GPU matrix multiplication...
Total number of blocks: 46610
Total number of threads: 11932160
GPU matrix multiplication took 299241542 nanoseconds

FLOPS: 7.68183e+10

Time in seconds: 0.150121

GPU GFLOPS of kernel only: 5.11711e+11

FLOPS: 7.68183e+10

Time in seconds: 0.299242

GPU GFLOPS with copy: 2.5671e+11
Comparing matrices...

Matrices are equal
```

## **Utilization and occupancy analysis:**

Full-occupancy and very high memory throughput and compute throughput

We then tested it with bigger blocks **32x32**

Result: 0 - 115 - MatrixMulKernel

Current 115 - MatrixMulKernel (148, 79, 1)x(32, 32, 1) 172.21 msecound 1E-YYYYYYYY 1E- 0 - NVIDIA GeForce RTX 3090 1.40 cycle/nsecond 8.6 [270519] task1

### GPU Speed Of Light Throughput

High-level overview of the throughput for compute and memory resources of the GPU. For each unit, the throughput reports the achieved percentage of utilization with respect to the theoretical maximum. Breakdowns show the throughput for each individual sub-metric of Compute and Memory to clearly identify the highest contributor.

	98.02 Duration [msecond]	172.21
Compute (SM) Throughput [%]	98.02 Duration [msecond]	172.21
Memory Throughput [%]	25.09 Elapsed Cycles [cycle]	240,229,333
L1/TEX Cache Throughput [%]	25.18 SM Active Cycles [cycle]	239,424,970.80
L2 Cache Throughput [%]	9.11 SM Frequency [cycle/nsecond]	1.40
DRAM Throughput [%]	6.74 DRAM Frequency [cycle/nsecond]	9.49

**High Throughput** The kernel is utilizing greater than 80.0% of the available compute or memory performance of the device. To further improve performance, work will likely need to be shifted from the most utilized to another unit. Start by analyzing workloads in the [Compute Workload Analysis](#) section.

### Launch Statistics

Summary of the configuration used to launch the kernel. The launch configuration defines the size of the kernel grid, the division of the grid into blocks, and the GPU resources needed to execute the kernel. Choosing an efficient launch configuration maximizes device utilization.

	11,692 Function Cache Configuration	cudaFuncCachePreferNone
Grid Size	11,692 Function Cache Configuration	cudaFuncCachePreferNone
Registers Per Thread [register/thread]	40 Static Shared Memory Per Block [byte/block]	0
Block Size	1,024 Dynamic Shared Memory Per Block [byte/block]	0
Threads [thread]	11,972,608 Driver Shared Memory Per Block [Kbyte/block]	1.02
Waves Per SM	142.59 Shared Memory Configuration Size [kbyte]	8.19

### Occupancy

Occupancy is the ratio of the number of active warps per multiprocessor to the maximum number of possible active warps. Another way to view occupancy is the percentage of the hardware's ability to process warps that is actively in use. Higher occupancy does not always result in higher performance, however, low occupancy always reduces the ability to hide latencies, resulting in overall performance degradation. Large discrepancies between the theoretical and the achieved occupancy during execution typically indicates highly imbalanced workloads.

	66.67 Block Limit Registers [block]	1
Theoretical Occupancy [%]	66.67 Block Limit Registers [block]	1
Theoretical Active Warps per SM [warp]	32 Block Limit Shared Mem [block]	8
Achieved Occupancy [%]	65.64 Block Limit Warps [block]	1
Achieved Active Warps Per SM [warp]	31.51 Block Limit SM [block]	16

**Occupancy Limiters** This kernel's theoretical occupancy (66.7%) is limited by the number of warps within each block. This kernel's theoretical occupancy (66.7%) is limited by the number of required registers. See the [CUDA Best Practices Guide](#) for more details on optimizing occupancy.

And the occupancy dropped by more than 33% as the register file was not enough for this many threads in a block.

You can see in the screenshots from the Nsight compute that despite a 33% drop in occupancy, the timing is still lower than the 16x16 by a 0.1 second.

## Task 2 for same matrices with 16x16 blocks

CPU matrix multiplication took 199626592983 nanoseconds (199.627 seconds)

**FLOPS: 7.68183e+10**  
**CPU GFLOPS: 3.8481e+08**

Total number of blocks: 46610

Total number of threads: 11932160

GPU matrix multiplication took 282432486 nanoseconds ( 0.2824 sec with copy and allocation)

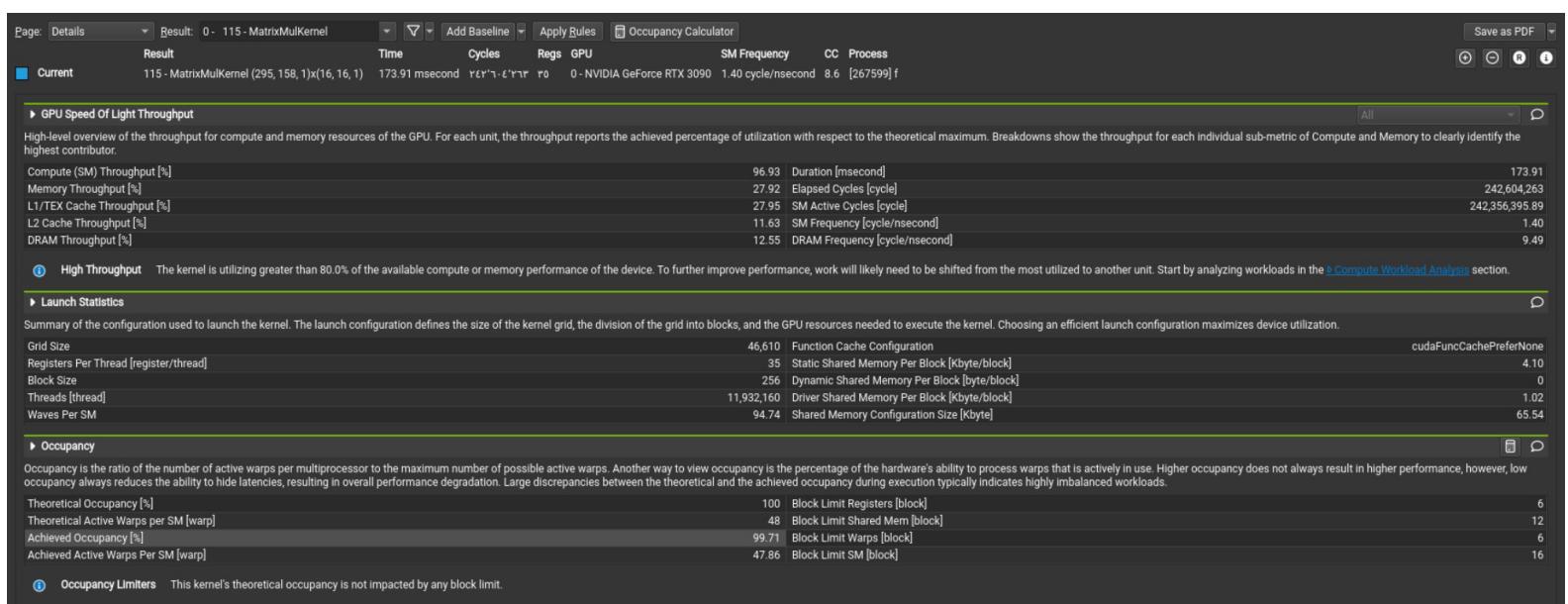
Kernel only Time in seconds: 0.143208

GPU GFLOPS of kernel only: **5.36411e+11**

GPU GFLOPS with copy: **2.71988e+11**

Speedup of Kernel over CPU is: **(1393.96 The fastest one we had)**

Speedup of kernel + allocation and copy is **706.81 over the CPU (fastest overall)**



```

● ➔ GPU ./gpu_assignment

Starting...
Dimensions of A: 2523x3232
Dimensions of B: 3232x4711
Starting CPU matrix multiplication...
CPU matrix multiplication took 199626592983 nanoseconds

FLOPS: 7.68183e+10

Time in seconds: 199.627

CPU GFLOPS: 3.8481e+08

Starting GPU matrix multiplication...
Total number of blocks: 46610
Total number of threads: 11932160

GPU matrix multiplication took 282432486 nanoseconds

FLOPS: 7.68183e+10

Time in seconds: 0.143208

GPU GFLOPS of kernel only: 5.36411e+11

FLOPS: 7.68183e+10

Time in seconds: 0.282432

GPU GFLOPS with copy: 2.71988e+11

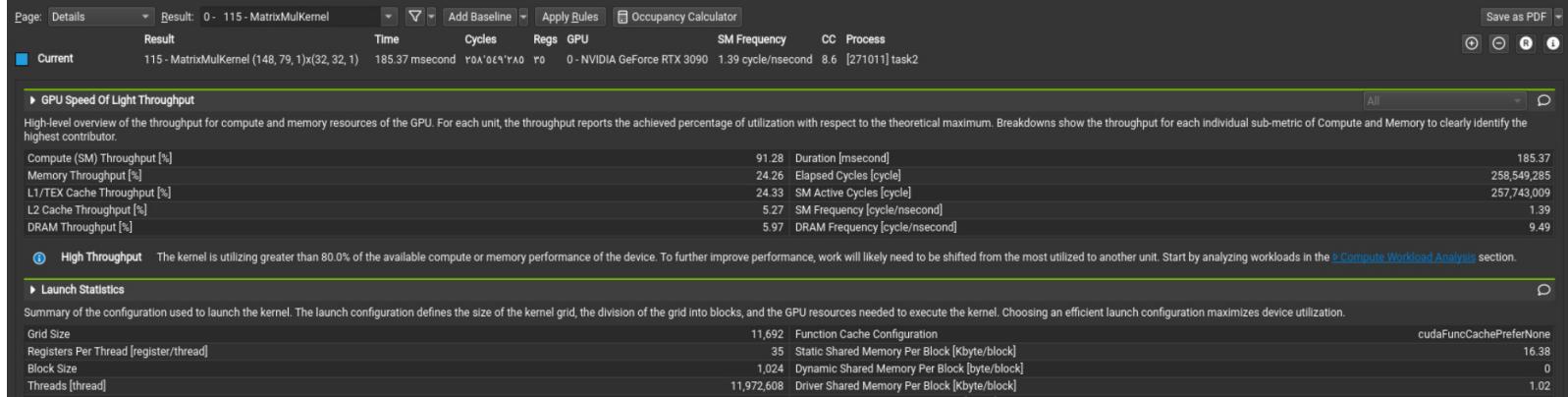
Comparing matrices...

Matrices are equal

```

You can see that this has the highest achieved occupancy of all kernels tested.  
 It also has one of the highest memory thorughputs of 28% and Compute throughput of 97%

We gave it another try at 32x32 blocks



**Which limited occupancy to 66.67 as shared memory was the limiting factor here unlike the previous test. This also resulted in around 91% Compute throughput which is significantly lower than previous implementations.**

```
Matrices are equal
● ➔ GPU ./gpu_assignment

Starting...
Dimensions of A: 2523x3232
Dimensions of B: 3232x4711
Starting CPU matrix multiplication...
CPU matrix multiplication took 193581901451 nanoseconds

FLOPS: 7.68183e+10

Time in seconds: 193.582

CPU GFLOPS: 3.8481e+08

Starting GPU matrix multiplication...
Total number of blocks: 11692
Total number of threads: 11972608

GPU matrix multiplication took 282432486 nanoseconds

FLOPS: 7.68183e+10

Time in seconds: 0.143208

GPU GFLOPS of kernel only: 5.36411e+11

FLOPS: 7.68183e+10

Time in seconds: 0.28243

GPU GFLOPS with copy: 2.71988e+11

Comparing matrices...

Matrices are equal
○ ➔ GPU
```

In the end, it was slower by 10 milliseconds as you can see in the Kernel duration in the upper right corner of the screenshot.

### **Additional notes:**

We wrote the input and output matrices to files instead of dumping them to the terminal for easier testing and debugging. So taking screenshots was not that useful as you can never see both inputs and outputs as well as the CPU and GPU results in one screenshot.