Computer Systems Organization and Architecture

Project 2 | Anvesh Muppeda

Contents

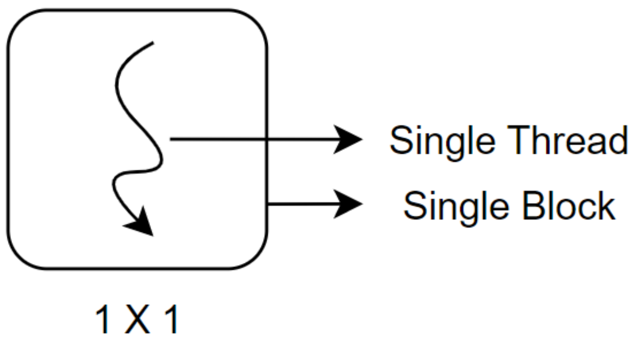
Execution Steps:

Created a **Makefile**, which will compile all versions of codes. And created a shell script(**run.sh**) that will execute all versions of codes with proper commands or we can compile and run them individually.

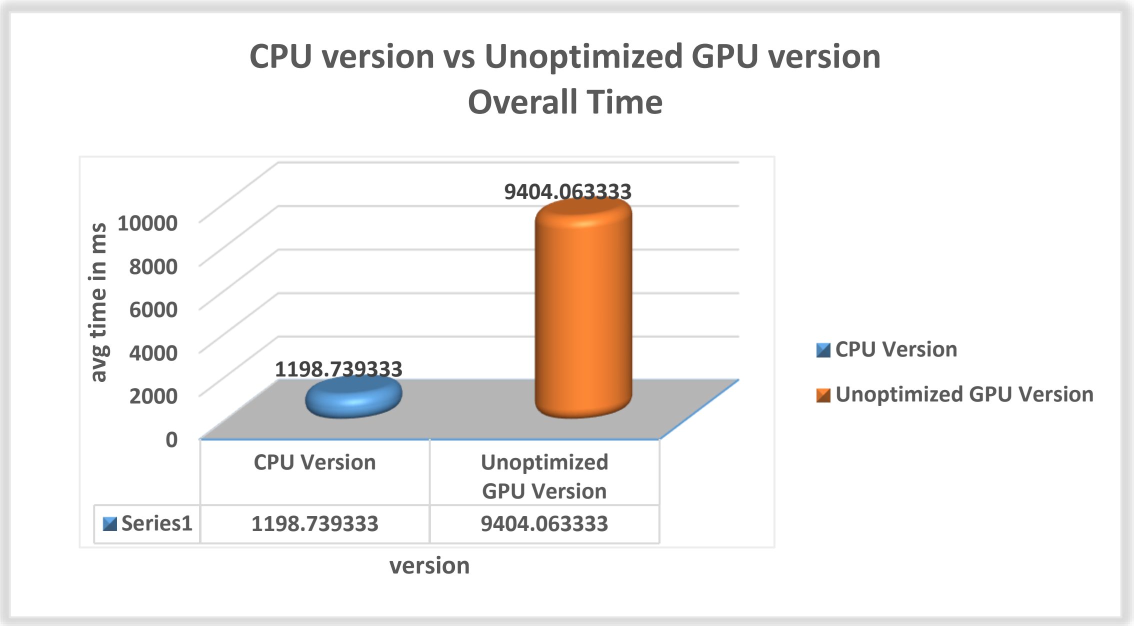
Part 1 (Unoptimized GPU version):

Here CPU version is optimal compared to the GPU version since the CPU version is executed faster than the GPU version. We are calling a single thread and a single block, which means we are not using the threads and blocks properly. We are initializing the matrices on Host and using those matrices on GPU with a single block and single thread, which means we are not efficiently using the GPU threads and blocks.

With the below diagram, we can define the Part 1 concept.



Based on the below graph we can determine the difference between the CPU version and current GPU version **performance**.



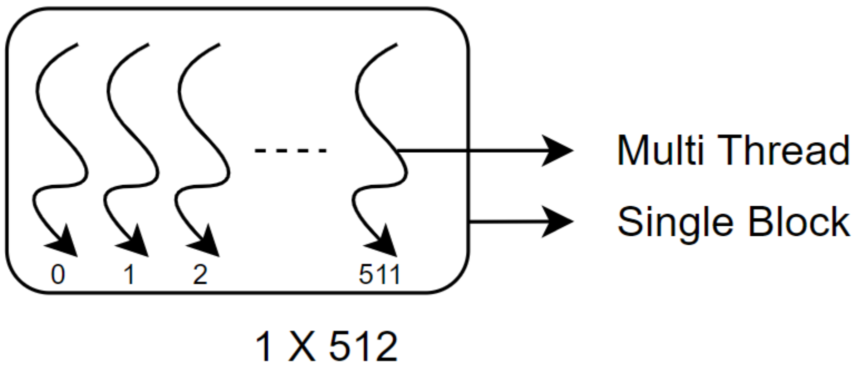
	CPU Version	Unoptimized GPU Version
Series1	1198.739333	9404.063333

From the GPU Activities gpu_Matrix_Multiplication kernel took 9.43318s total in execution.

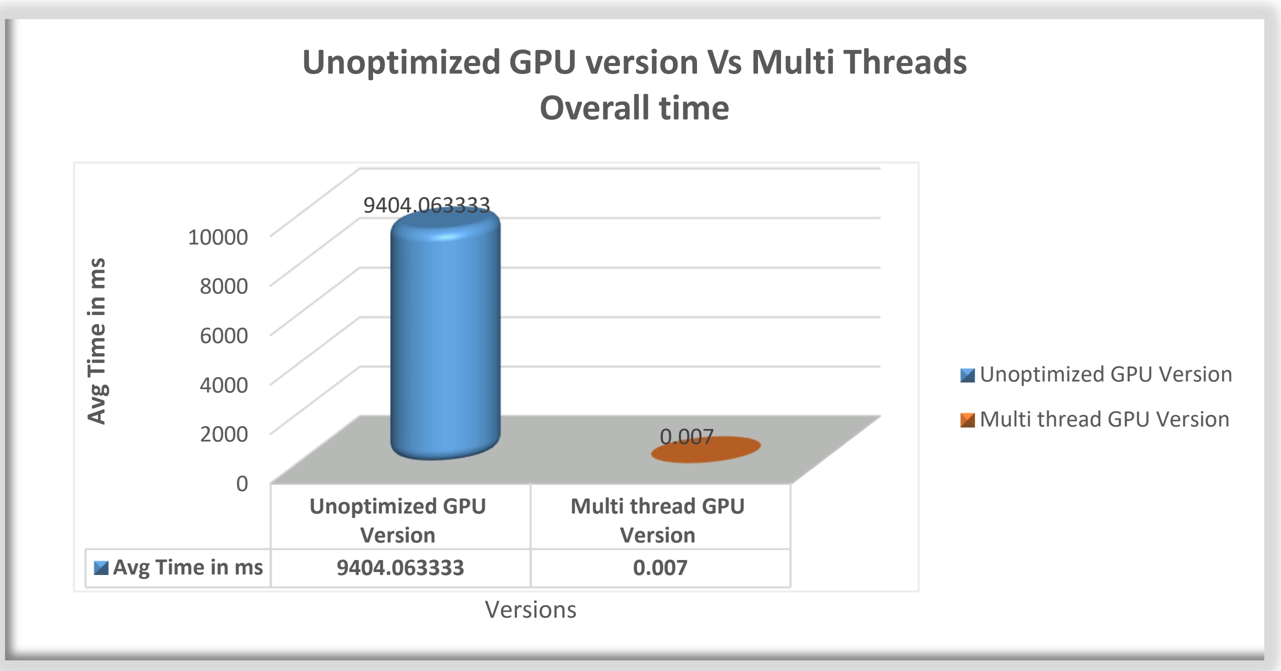
Part 2 (Multiple Threads):

In this section, we are using multiple threads (512) with a single block. Hence we are calling the kernel 512 times onto the GPU device, based on this we can get a better execution time overall and also a much better execution time on the matrix multiplication kernel.

With the below diagram, we can define the part 2 concept.



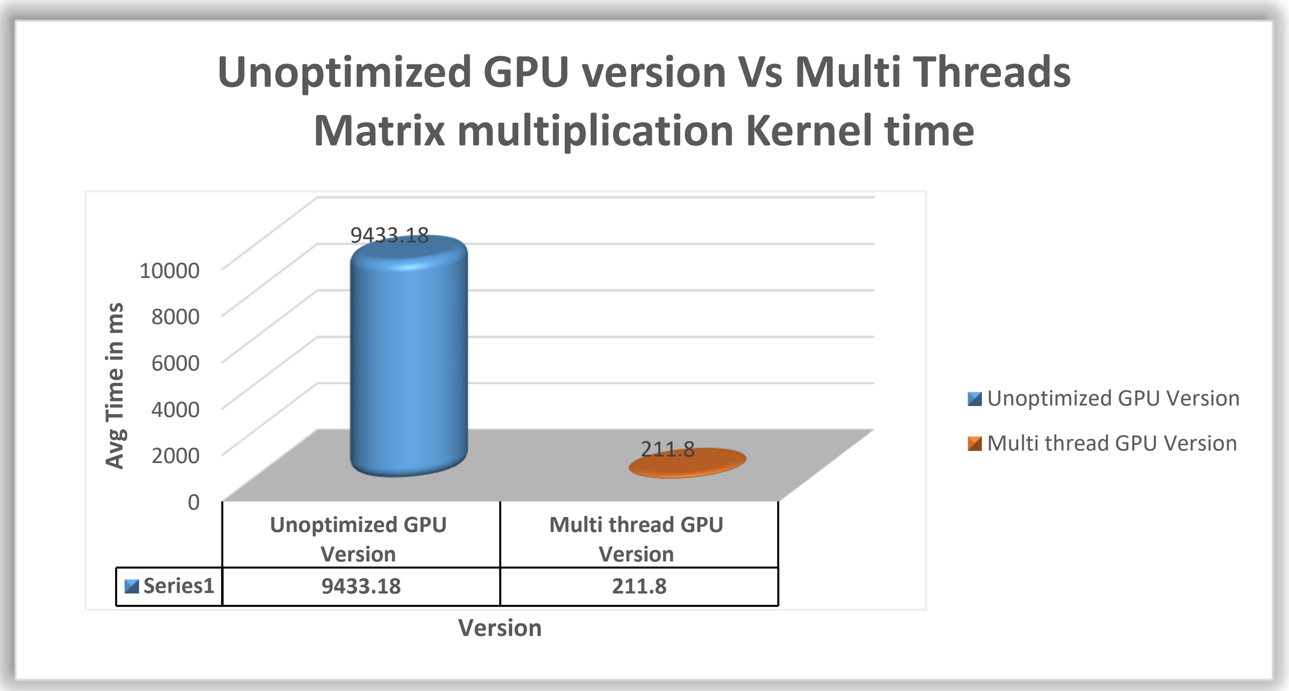
Also, gpu_Matrix_Multiplication kernel execution time was reduced to **211.80ms** from **9.43318s**



	Unoptimized GPU Version	Multi thread GPU Version
Avg Time in ms	9404.063333	0.007

Matrix Multiplication Kernel Performance:

Which increased the performance and decreased the time of execution.



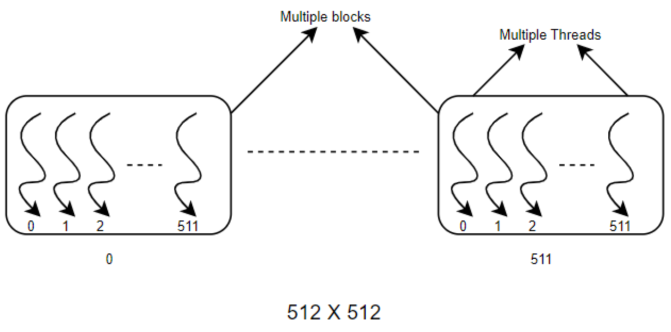
	Unoptimized GPU Version	Multi thread GPU Version
Series1	9433.18	211.8

Part 3 (Multiple Blocks):

Here we are using multiple blocks and multiple threads. Compared to the previous version here we are introducing the **multiple blocks** as well along with the multiple threads. So we are dividing the matrix multiplication operations into multiple threads based on the blocks and threads. With this, we can get better execution and performance on overall time and matrix multiplication kernel time.

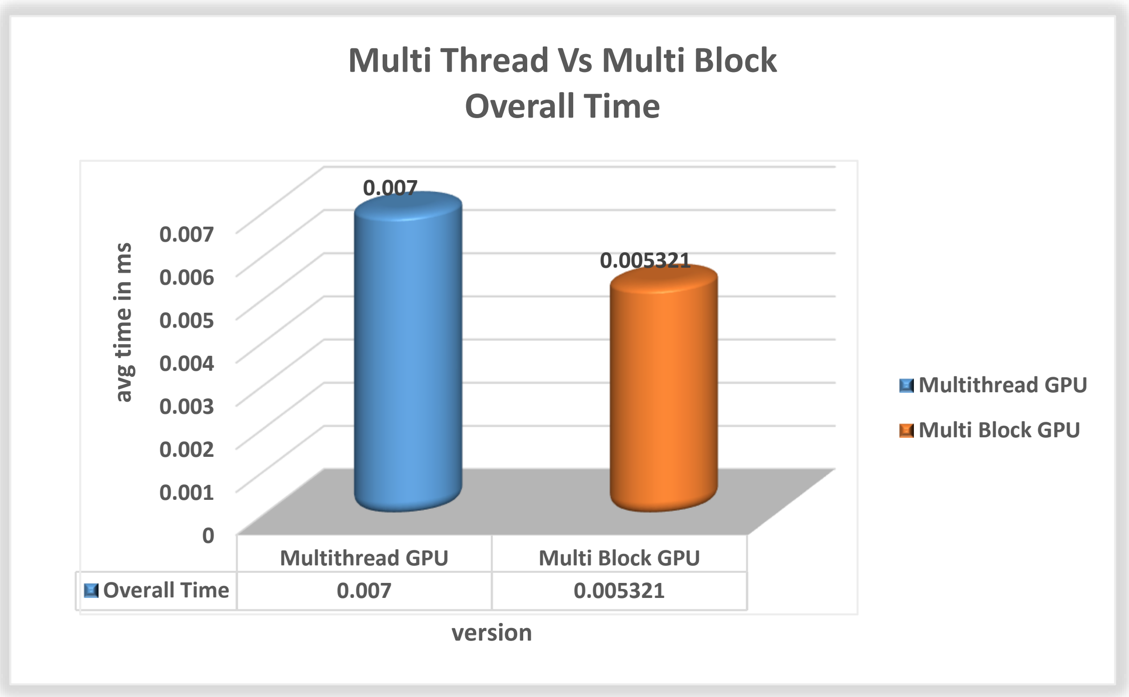
With this, we can see the efficient execution time on the matrix multiplication kernel.

With the below diagram, we can define the part 3 concept.



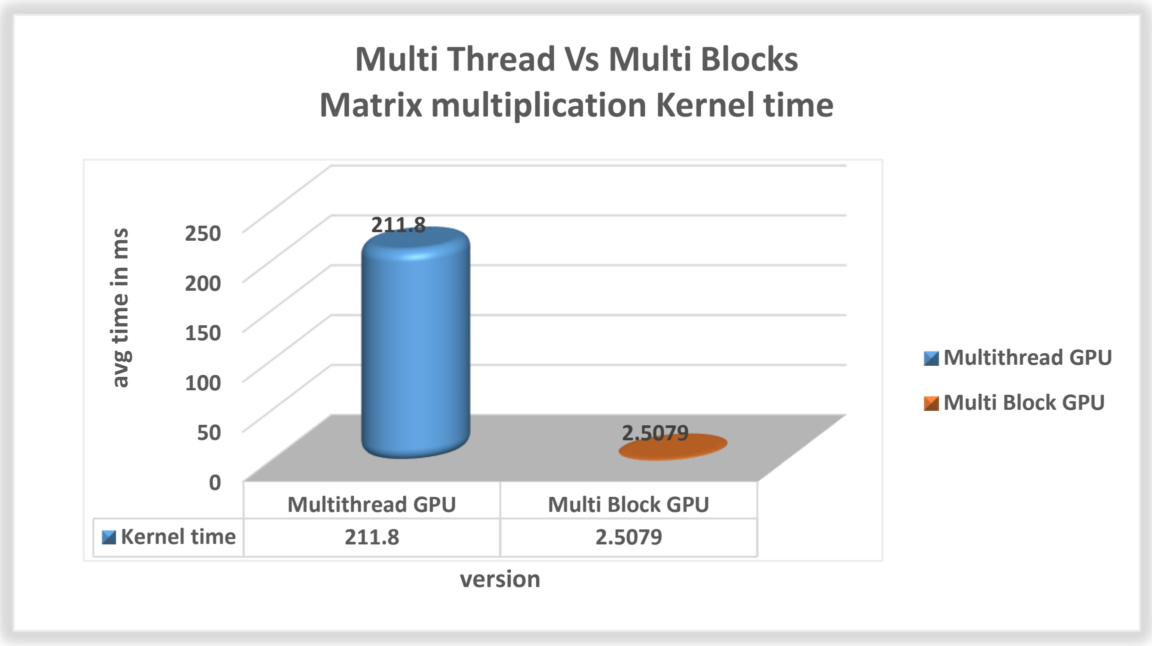
Example 1:

Overall execution Performance:



	Multithread GPU	Multi Block GPU
Overall Time	0.007	0.005321

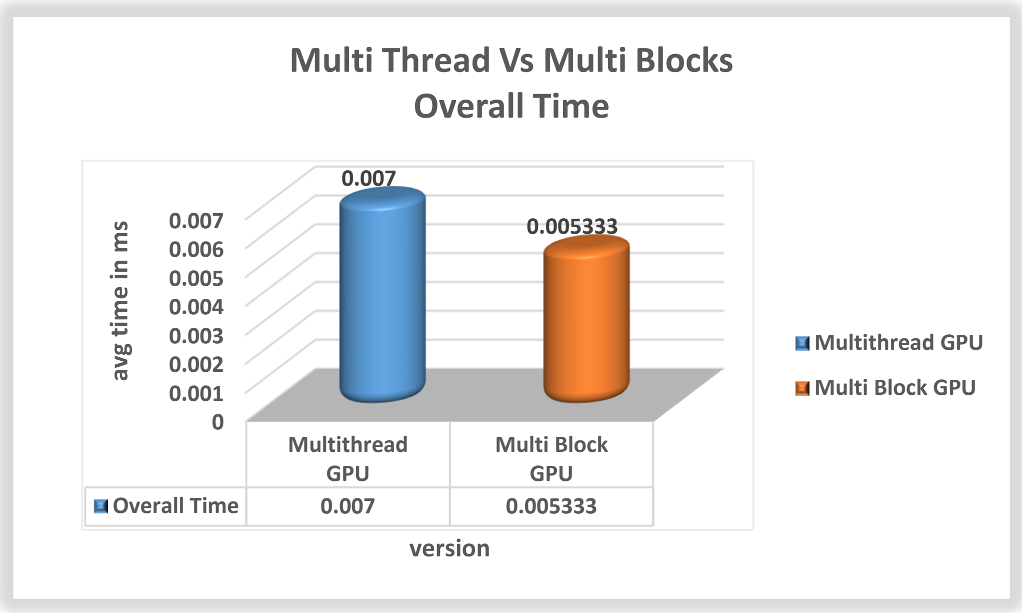
Matrix Multiplication Kernel Performance:



	Multithread GPU	Multi Block GPU
Kernel time	211.8	2.5079

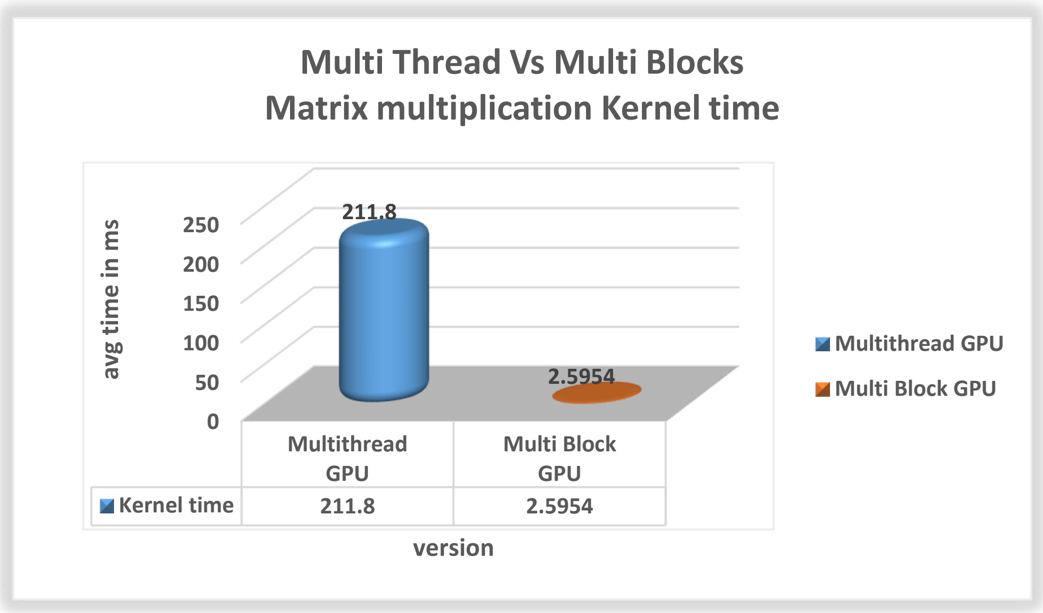
Example 2:

Overall execution time (Performance):



	Multithread GPU	Multi Block GPU
Overall Time	0.007	0.005333

Matrix Multiplication performance:



Part 4 (Optimize and Prefetch):

Here we are using the same as previously i.e. multi threads and multi blocks But here we are optimizing the number of threads and blocks to get better performance (i.e. **256 X 256**). With the optimized version, we can get better performance on the overall performance and also on matrix multiplication kernel execution time.

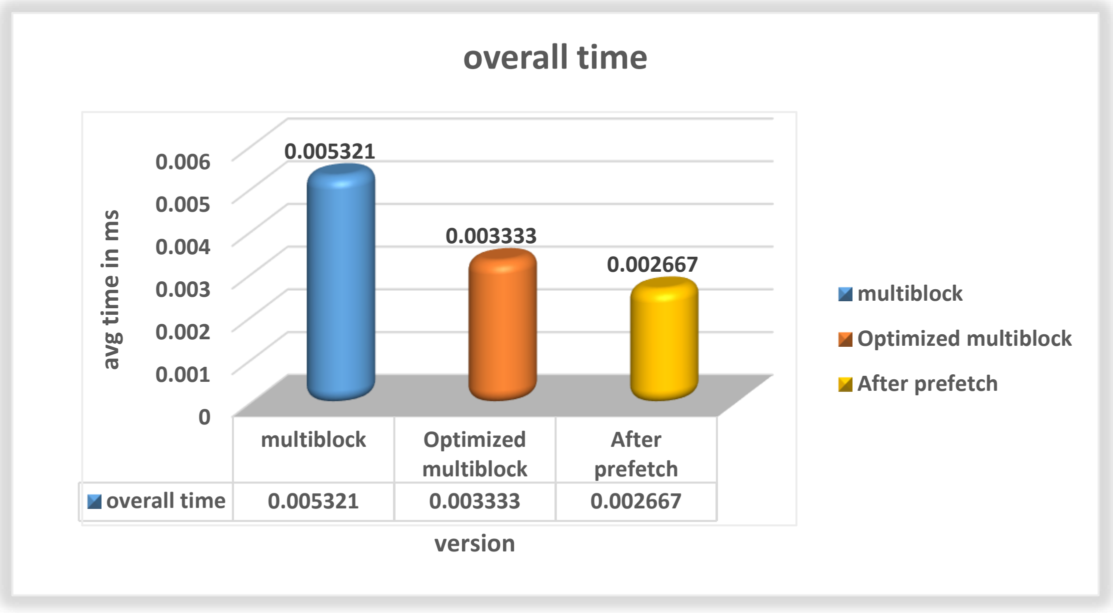
Page faults:

With the multi threads and multi blocks, we can get a better execution time but it affects the number of page faults since we are defining the initialization of matrices on the host and we are using these on GPU. In this process, the number of page faults will increase, in order to decrease or avoid the page faults we are defining the **initialization kernel on to the GPU device**. Here the page faults in the host initialization concept i.e. in previous versions are **24** but after GPU device initialization the number of page faults is reduced to **6**.

Prefetch:

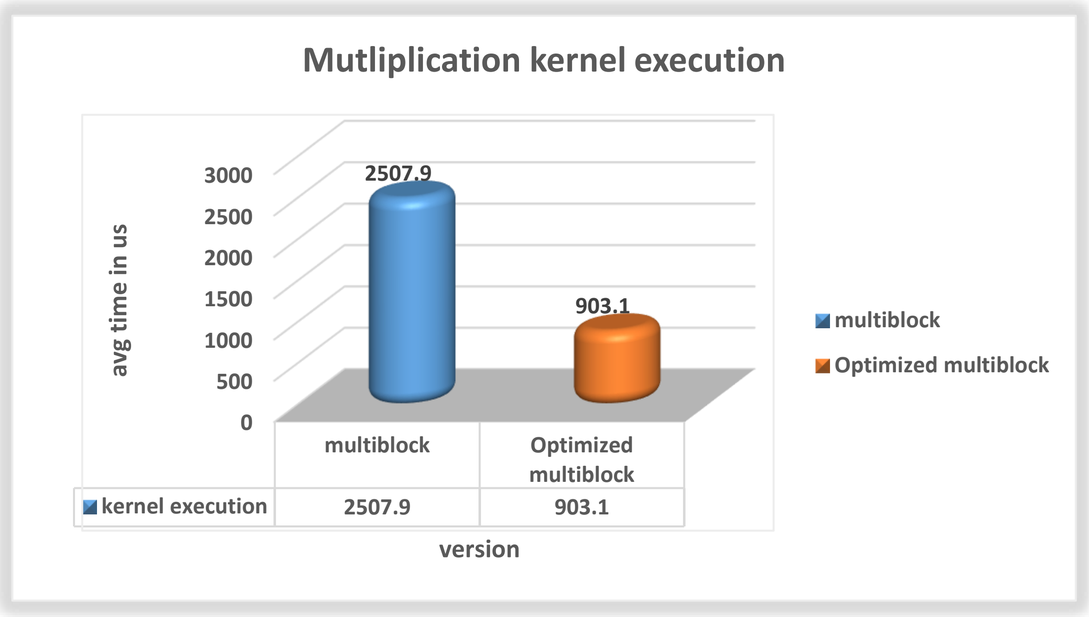
So with the GPU initialization, we can get a less number of page faults but still, we are getting more time on to the initialization kernel execution. In order to overcome this, we are using the prefetch concept. By using the **prefetch we can decrease the initialization kernel execution time**. After adding the prefetch concept, our initialization kernel execution decreased a lot, and performance increased.

Overall performance:



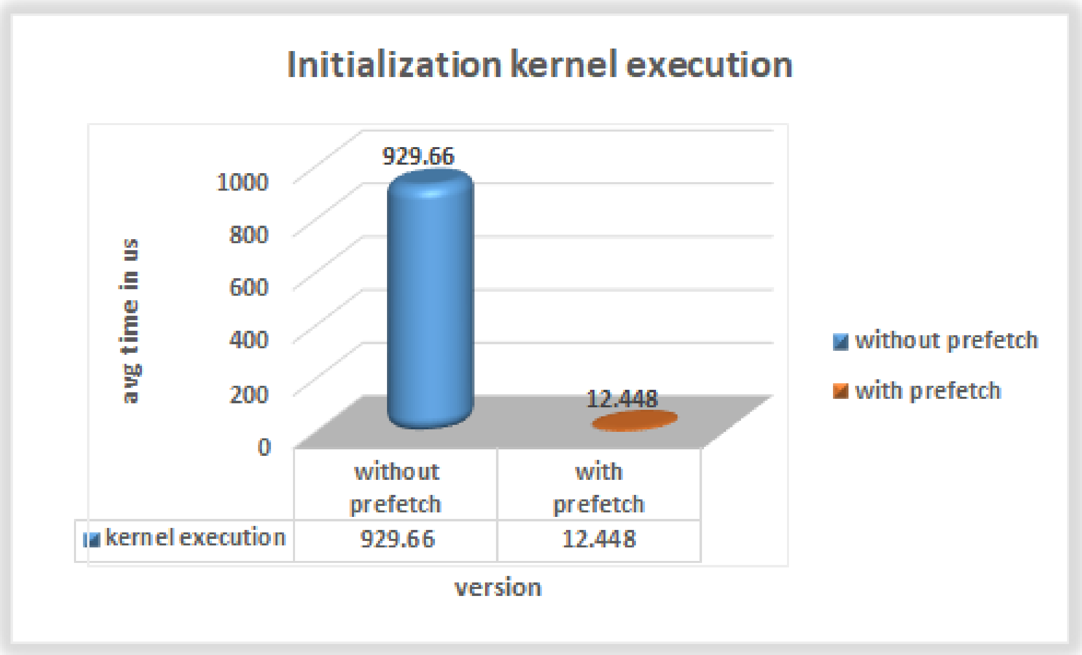
	multiblock	Optimized multiblock	After prefetch
overall time	0.005321	0.003333	0.002667

Matrix Multiplication Kernel Performance:

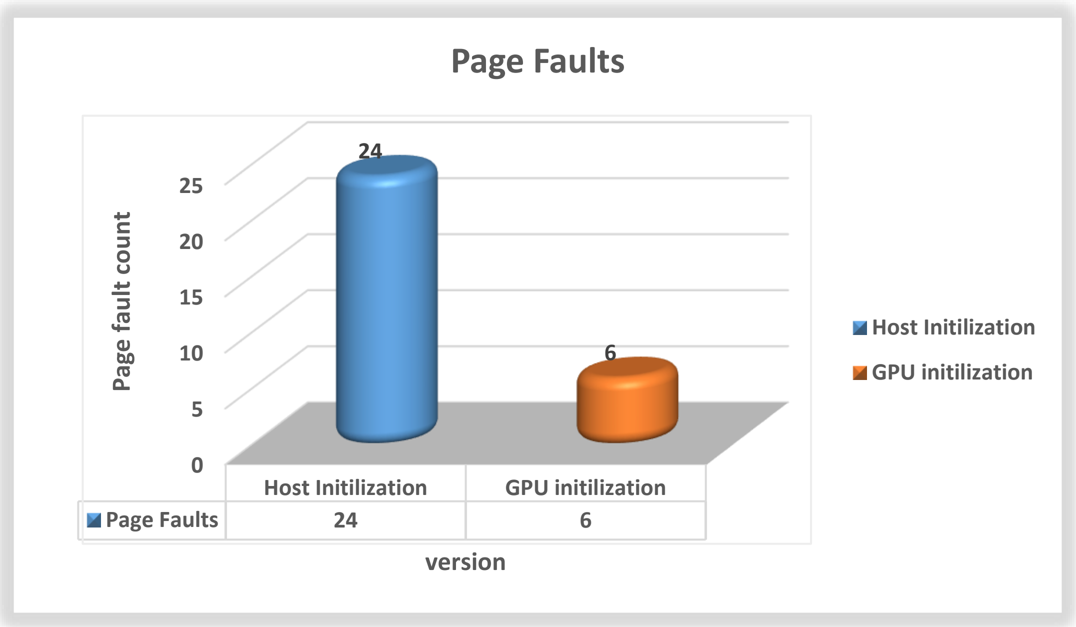


	multiblock	Optimized multiblock
kernel execution	2507.9	903.1

Initialization Kernel Performance:



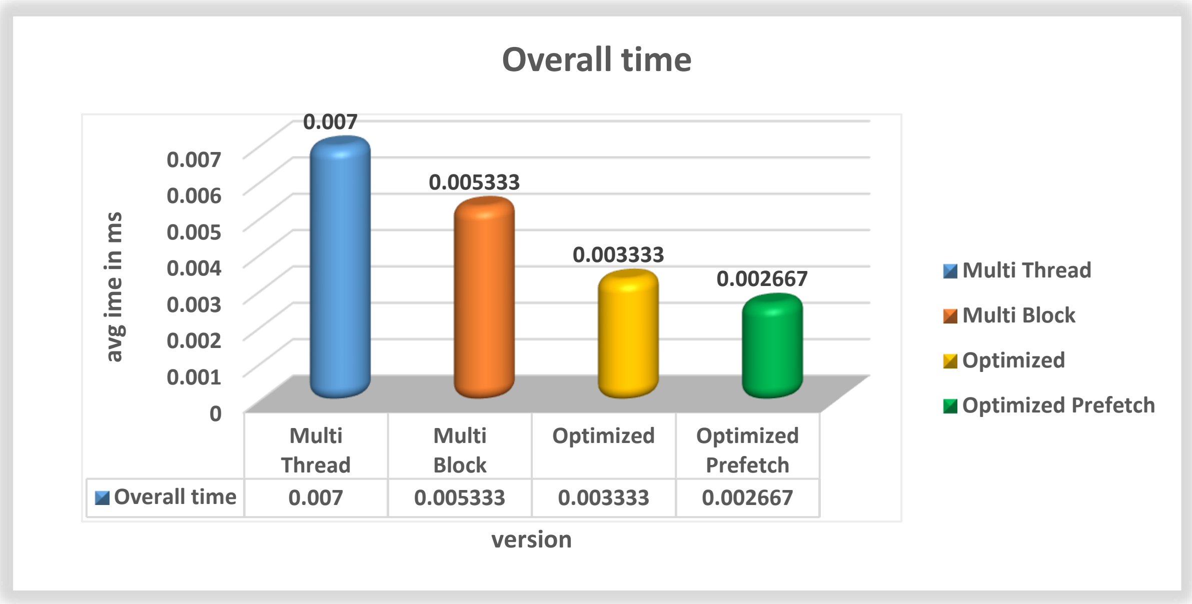
Page faults:



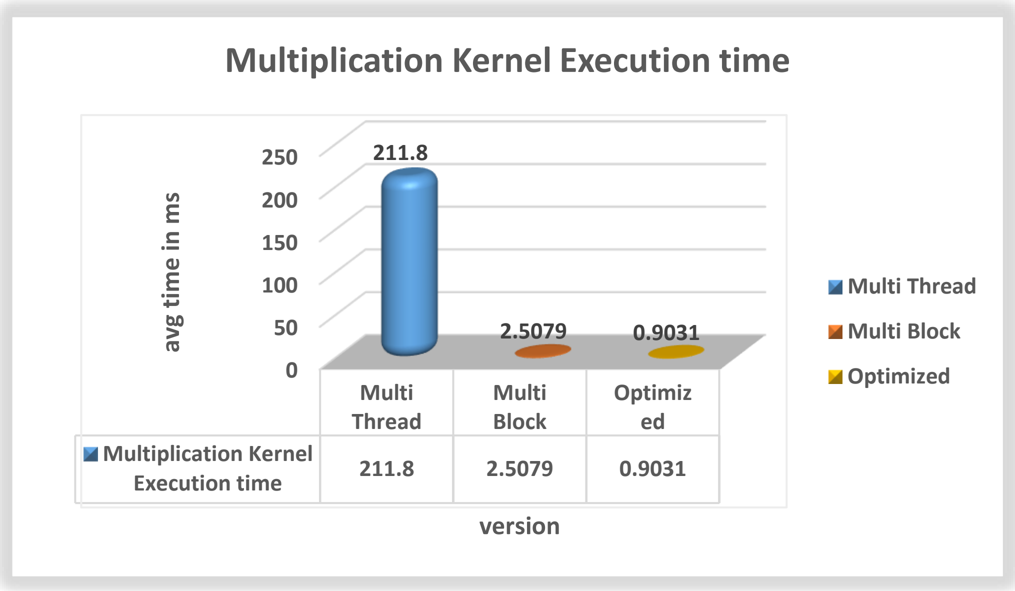
Overall Summary:

GPU version is better than the CPU version when it comes to huge number of independent operations. In this project through the various versions we can observe the performance changes in overall kernels. In the last version we used all available options like optimized threads and blocks and also initialization on kernel in CUDA to get the better performance.

Overall Performance:



Matrix Multiplication Performance:


Multiplication Kernel Execution time
avg time in ms
Multi Thread
Multi Block
Optimiz ed
Multiplication Kernel Execution time
version
Multi Thread
Multi Block
Optimized