# Technical Report

Omar Sanseviero Güezmes
August 2018

**Summary**
*This report explores solving the problem of matrix multiplication using CPU and GPU with different configurations. Using GPU leads to a clear improvement in the performance. The report compares empirical results (execution time) of the different configurations.*

## 1. Introduction

CUDA has an interesting programming model that boosts the performance in problems with lots of data. A simple problem that is data-intensive is high-dimensional matrix multiplication. After a certain point, using CPU, even with threads, is not a viable option to multiply matrices. CUDA gives developers an abstraction of the GPU architecture, making it easy to transfer data from the host (CPU) to the device (GPU), which has an internal memory. The device has a grid of blocks and each block has threads. Changing the setup of the grid and the blocks has a direct impact in the performance. Because of this, we'll explore the results of different-size matrices with the following configurations: CPU, CPU with threads using OpenMP, GPU using a 1D grid, GPU using a 2D grid, and GPU using a 2D grid with 2D blocks.

## 2. Development and results

As mentioned in the introduction, CUDA gives control to the developers. When a kernel function is called from the host, the device obtains execution and automatically generates the threads and blocks as specified in the host. Having access to the block and grid dimensions gives opportunity for optimization.
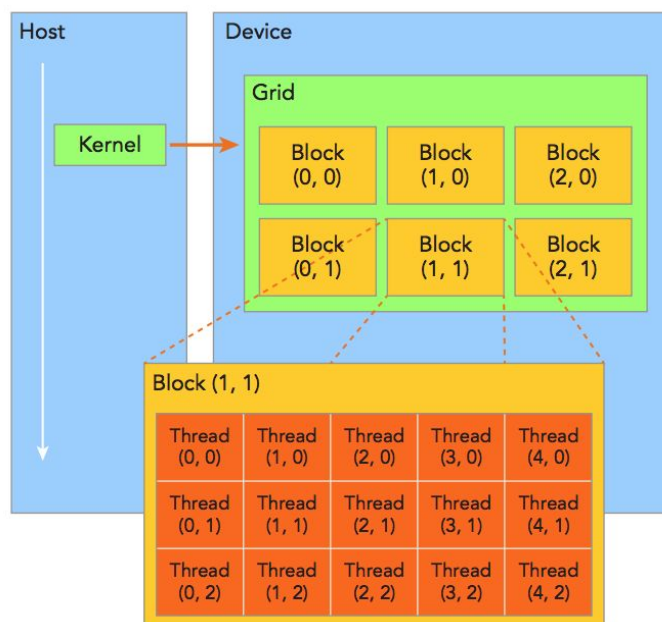
To test the configurations, we'll use matrices stored linearly. Doing this allows us to easily map threads to data elements in the matrices. We measure time with 3x3, 1000x1000, 2000x2000 and 4000x4000 matrices with natural numbers using row-major order. Time is measured using chrono *high_resolution_clock*[1].

The programs are being run using GeForce GTX 670. It has 1344 CUDA cores and 7 multiprocessors. The GPU max clock rate is 0.98 GHz. The memory clock rate is 3004 Mhz. For CPU the programs are being run in an Intel i7-4770 Processor, which has 4 cores. The clock speed is 3.4 GHz.

The following tables show 5 trials for each configuration. CPU was unable to run 2000x2000 matrix multiplication in an acceptable time. All times in the tables are measured in milliseconds. Changing the number of threads in the GPU implementations did not have any representative impact in performance.

| CPU | | | |
|---|---|---|---|
| | **3x3** | **1000x1000** | **2000x2000** |
| **Trial 1** | 0.001302 | 5643.34 | 81837.3 |
| **Trial 2** | 0.001257 | 5902.61 | 74183.4 |
| **Trial 3** | 0.00128 | 7837.91 | 74752.7 |
| **Trial 4** | 0.001618 | 9388.52 | 75171.9 |
| **Trial 5** | 0.001015 | 6082.07 | 75222.4 |
| **Average** | **0.0012944** | **6970.89** | **76233.54** |

| CPU with threads | | | |
|---|---|---|---|
| | **3x3** | **1000x1000** | **2000x2000** |
| **Trial 1** | 0.000408 | 6315.75 | 72933.7 |
| **Trial 2** | 0.00121 | 7950.09 | 74614.7 |
| **Trial 3** | 0.001315 | 6904.19 | 73207.2 |
| **Trial 4** | 0.00107 | 6983.94 | 75365.1 |

---

[1] https://en.cppreference.com/w/cpp/chrono/high_resolution_clock

| | | | |
|---|---|---|---|
| Trial 5 | 0.000986 | 5894.8 | 74223.6 |
| **Average** | **0.0009978** | **6809.754** | **74068.86** |

| GPU 1D grid, 1D block, 128 threads per block | | | |
|---|---|---|---|
| | **3x3** | **1000x1000**<br>**8 blocks** | **2000x2000**<br>**16 blocks** | **4000x4000**<br>**32 blocks** |
| **Trial 1** | 0.036468 | 533.889709 | 1980.140259 | 7485.441895 |
| **Trial 2** | 0.029224 | 489.622406 | 1933.500244 | 8741.581055 |
| **Trial 3** | 0.039797 | 487.324371 | 1932.235107 | 7744.794922 |
| **Trial 4** | 0.040914 | 488.069641 | 1932.458008 | 7776.899414 |
| **Trial 5** | 0.041869 | 536.176270 | 1933.899658 | 8756.853516 |
| **Average** | **0.0376544** | **507.0164794** | **1942.4466552** | **8101.1141604** |

| GPU 2D grid, 1D block, 128 threads per block | | | |
|---|---|---|---|
| | **3x3**<br>1x3 grid | **1000x1000**<br>8x1000 grid | **2000x2000**<br>16x2000 grid | **4000x4000**<br>32x4000 grid |
| **Trial 1** | 0.033022 | 63.710728 | 498.139618 | 3831.066650 |
| **Trial 2** | 0.042266 | 64.600227 | 468.093292 | 3799.276367 |
| **Trial 3** | 0.024313 | 63.881649 | 468.092377 | 3807.784424 |
| **Trial 4** | 0.041660 | 63.774837 | 467.573517 | 3802.318359 |
| **Trial 5** | 0.042253 | 62.991325 | 466.071503 | 3815.585449 |
| **Average** | **0.0367028** | **63.7917532** | **473.5940614** | **3811.2062498** |

| GPU 2D grid, 2D block, 32x32 block size | | | |
|---|---|---|---|
| | **3x3**<br>1x1 grid | **1000x1000**<br>32x32 grid | **2000x2000**<br>63x63 grid | **4000x4000**<br>125x125 grid |

| | | | | |
|---|---|---|---|---|
| **Trial 1** | 0.030596 | 58.653423 | 449.625061 | 3166.145996 |
| **Trial 2** | 0.040654 | 62.907055 | 413.866425 | 3147.636963 |
| **Trial 3** | 0.037539 | 61.644348 | 399.948914 | 3155.324951 |
| **Trial 4** | 0.035190 | 62.770149 | 405.283752 | 3163.276855 |
| **Trial 5** | 0.020625 | 61.128883 | 403.779327 | 3165.669434 |
| **Average** | **0.0329208** | **61.4207716** | **414.5006958** | **3159.6108398** |

# 3. Analysis

The following table summarizes the results from the previous section.

| | **3x3** | **1000x1000** | **2000x2000** | **4000x4000** |
|---|---|---|---|---|
| **CPU** | 0.0012944 | 6970.89 | 76233.54 | |
| **CPU + OpenMP** | 0.0009978 | 6809.754 | 74068.86 | |
| **GPU 1D** | 0.0376544 | 507.0164794 | 1942.4466552 | 8101.1141604 |
| **GPU 2D 1D** | 0.0367028 | 63.7917532 | 473.5940614 | 3811.2062498 |
| **GPU 2D 2D** | 0.0329208 | 61.4207716 | 414.5006958 | 3159.6108398 |

## Matrix multiplication times
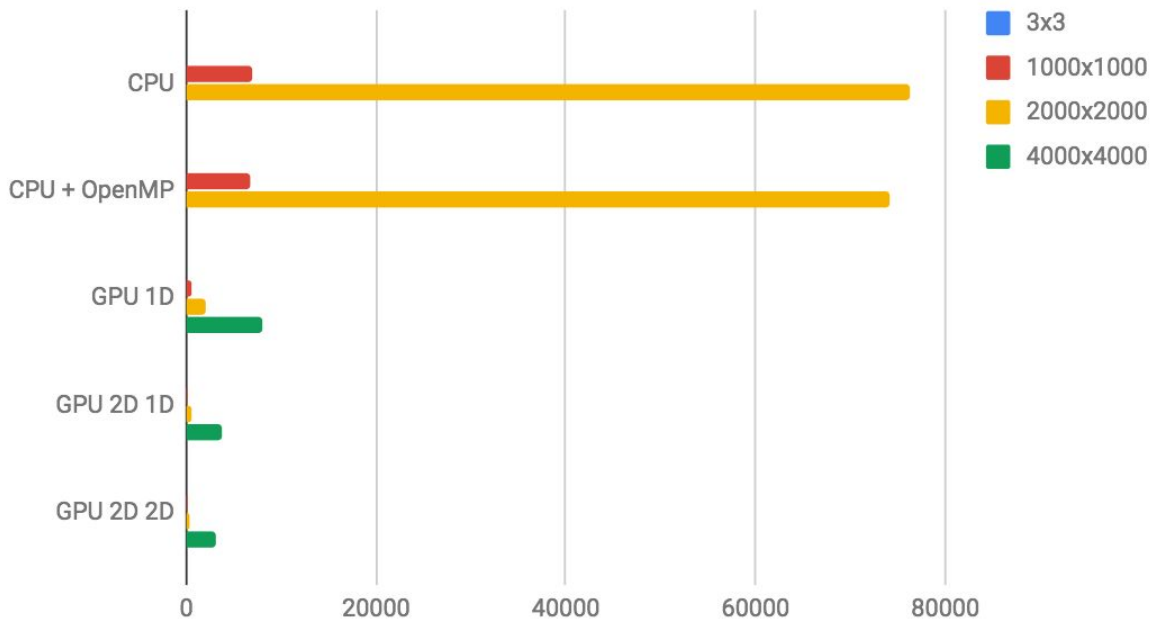


We obtain the best results using 2D grid with 2D array of threads per block. The difference between 2D 1D and 2D 2D is small, but still significant. The difference between CPU and CPU with threads is really small and not necessarily significant.

# 4. Conclusions

Using GPU takes longer to configure than CPU, as can be noted in small matrices (eg 3x3), but presents a huge boost in running time. There's a lot of variation in the time. This can be due to many factors, causing the results not to be consistent and have smaller statistical significance.

Changing the number of threads per block in the 3 experiments with GPU showed no direct impact in running time. This is because, at the end, we are always using the same number of threads: N. If a matrix is 1000x1000, the program checks that the thread is smaller than 1000 (ix < n). Because of this, even if we have more threads, the algorithm makes no use of them. I expected a better performance of CPU with threads in comparison to the CPU, but the results were pretty much the same when comparing to the GPU.