2018

EE 5123

Computer Architecture (Spring 2018)

GPU Benchmarking using CUDA framework (Nvidia)

BY

GOPAL VISHWAKARMA

CONTENTS

AIM OF THE PROJECT

The project evaluates performance of CPU & GPU and time taken to execute a code on each processor. In this, we will be giving huge array of numbers and compute it on CPU as well as on GPU. And we can test the time required to process it, also we will be testing this benchmarking program on different GPU since we have two PC that has different versions of Nvidia GPU.

INTRODUCTION

1.1 What are Graphics Processing Units?

A Graphics Processing Unit (GPU) is a microprocessor that has been designed specifically for the processing of 3D graphics. The processor is built with integrated transform, lighting, triangle setup/clipping, and rendering engines, capable of handling millions of math-intensive processes per second. GPUs form the heart of modern graphics cards, relieving the CPU (central processing units) of much of the graphics processing load. GPUs allow products such as desktop PCs, portable computers, and game consoles to process real-time 3D graphics that only a few years ago were only available on high-end workstations.

Used primarily for 3-D applications, a graphics processing unit is a single-chip processor that creates lighting effects and transforms objects every time a 3-D scene is redrawn. These are mathematically-intensive tasks, which otherwise, would put quite a strain on the CPU. Lifting this burden from the CPU frees up cycles that can be used for other jobs.

However, the GPU is not just for playing 3D-intense videogames or for those who create graphics (sometimes referred to as graphics rendering or content-creation) but is a crucial component that is critical to the PC's overall system speed. In order to fully appreciate the graphics card's role, it must first be understood.

Many synonyms exist for Graphics Processing Unit in which the popular one being the graphics card. It’s also known as a video card, video accelerator, video adapter, video board, graphics accelerator, or graphics adapter.

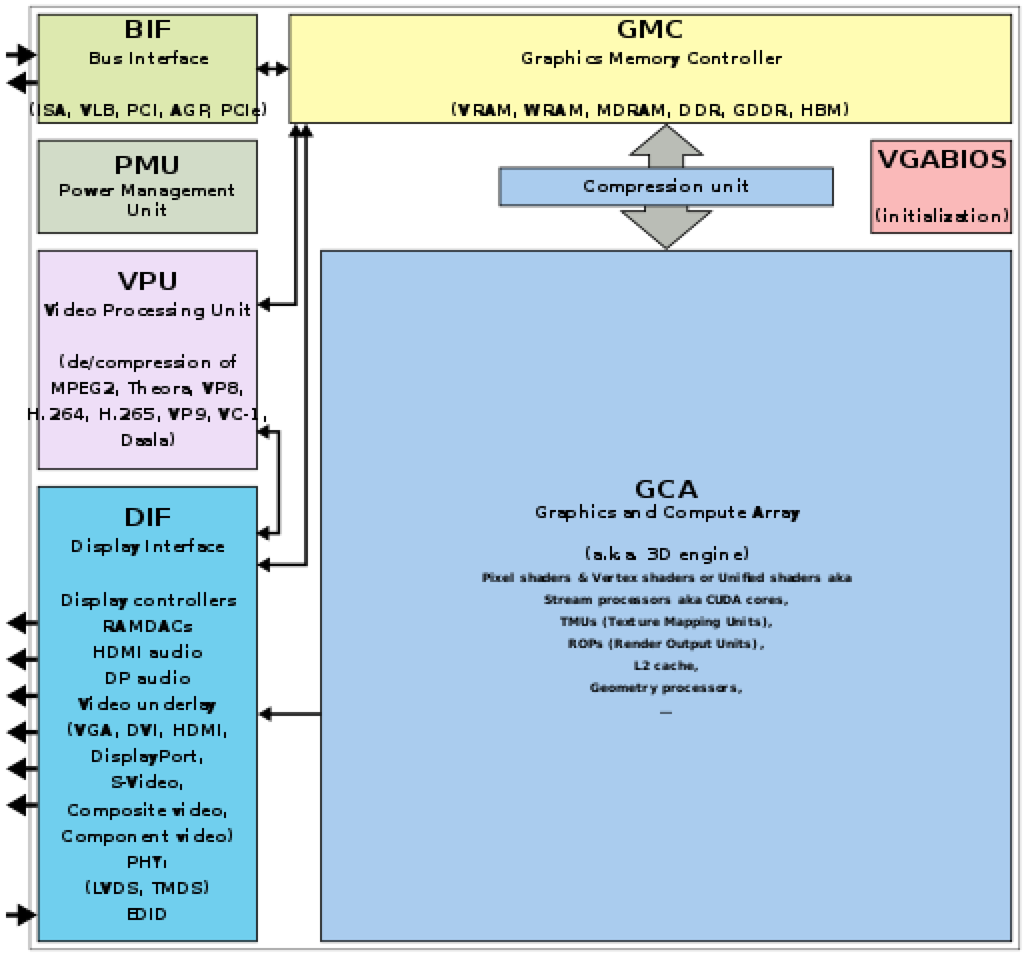

BIF
Bus Interface
(ISA, VLB, PCI, AGP, PCIe)
GMC
Graphics Memory Controller
(VRAM, WRAM, MDRAM, DDR, GDDR, HBM)
PMU
Power Management Unit
Compression unit
VGABIOS
(initialization)
VPU
Video Processing Unit
(de/compression of MPEG2, Theora, VP8, H.264, H.265, VP9, VC-1, Daala)
DIF
Display Interface
Display controllers
RAMDACs
HDMI audio
DP audio
Video underlay
(VGA, DVI, HDMI, DisplayPort, S-Video, Composite video, Component video)
PHYs
(LVDS, TMDS)
EDID
GCA
Graphics and Compute Array
(a.k.a. 3D engine)
Pixel shaders & Vertex shaders or Unified shaders aka Stream processors aka CUDA cores,
TMUs (Texture Mapping Units),
ROPs (Render Output Units),
L2 cache,
Geometry processors,
...

Fig.1 Components of a GPU

1.2 What is CUDA Programming Model?

CUDA is a parallel computing platform and application programming interface (API) model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit (GPU) for general purpose processing – an approach termed GPGPU (General-Purpose computing on Graphics Processing Units). The CUDA platform is a software layer that gives direct access to the GPU's virtual instruction set and parallel computational elements, for the execution of compute kernels.

The CUDA platform is designed to work with programming languages such as C, C++, and Fortran. This accessibility makes it easier for specialists in parallel programming to use GPU resources, in contrast to prior APIs like Direct3D and OpenGL, which required advanced skills in graphics programming. Also, CUDA supports programming frameworks such as OpenACC and OpenCL.

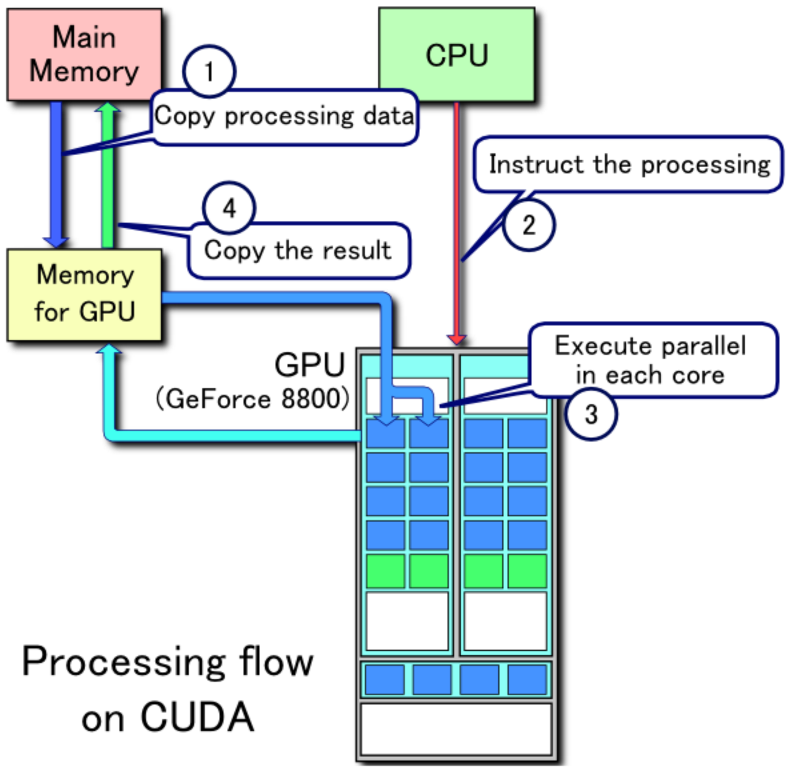


Fig.2 Simple Processing Flow of CUDA

1.3 What is HPP (Heterogeneous parallel programming)?

Heterogeneous Parallel Programming is about making the Graphics Card Processor (GPU) help CPU to do the heavy jobs in comparatively very much less time, for which the CPU, working alone might have taken much more time.

In essence, it boosts system performance dramatically i.e. exponentially.

Usually GPU's are made up of hundreds of core nearly same as we use quad core (4 cores) or dual core (2 cores) in PC. Generally, inside PC, operating system creates software threads that is managed by CPU scheduling algorithm. However, GPU's have hundreds of powerful cores and which are hardware threads that can be controlled by CPU. Previously, GPU are made for only for graphical application like PC games, animation, movies. But Nvidia added their hardware kit inside GPU so that we can use those cores for general computation.

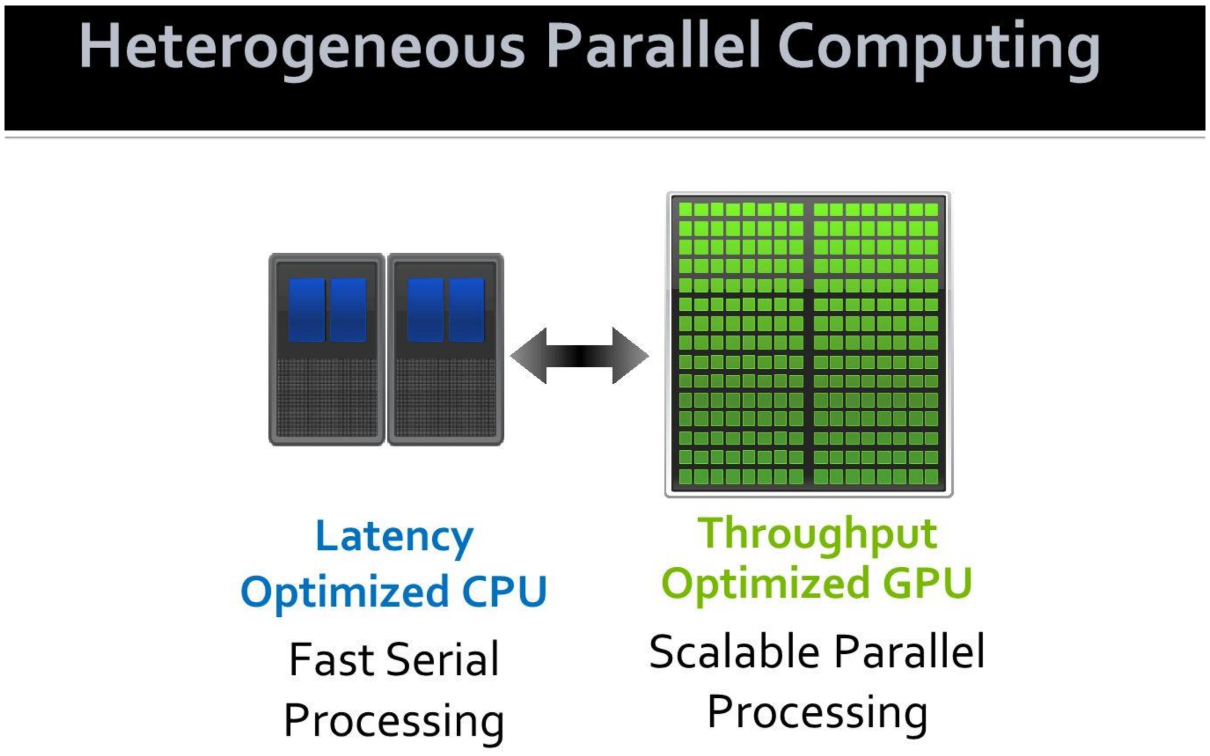


Fig.3 Heterogeneous parallel computing

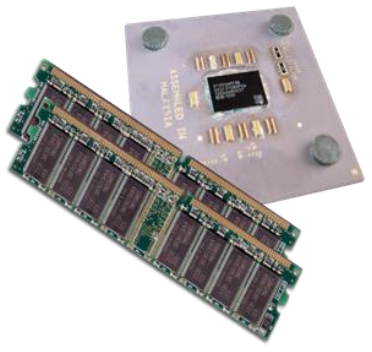


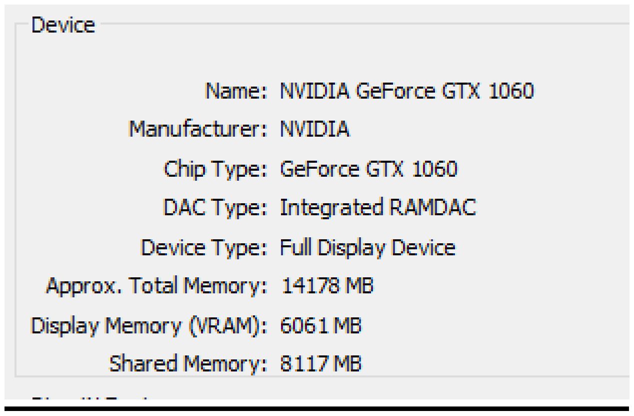
Fig.4 The CPU and its memory (host memory)

Fig.5 The GPU and its memory (device memory)

APPROACH OF THE PROJECT

The project is divided into three segments they are as follow,

1. Getting GPU device properties using CUDA API's.
2. Addition of a vector array (CPU as well as GPU).
3. Multiplication of 2D array.



Name:	NVIDIA GeForce GTX 1060
Manufacturer:	NVIDIA
Chip Type:	GeForce GTX 1060
DAC Type:	Integrated RAMDAC
Device Type:	Full Display Device
Approx. Total Memory:	14178 MB
Display Memory (VRAM):	6061 MB
Shared Memory:	8117 MB

Fig.6 GPU details

EXECUTION OF THE PROJECT

Firstly, the program needs to be compiled in the Nvidia CUDA compiler using the command,

After that the program creates a .exe file which is ready for use.

Nvcc (Nvidia CUDA compiler)

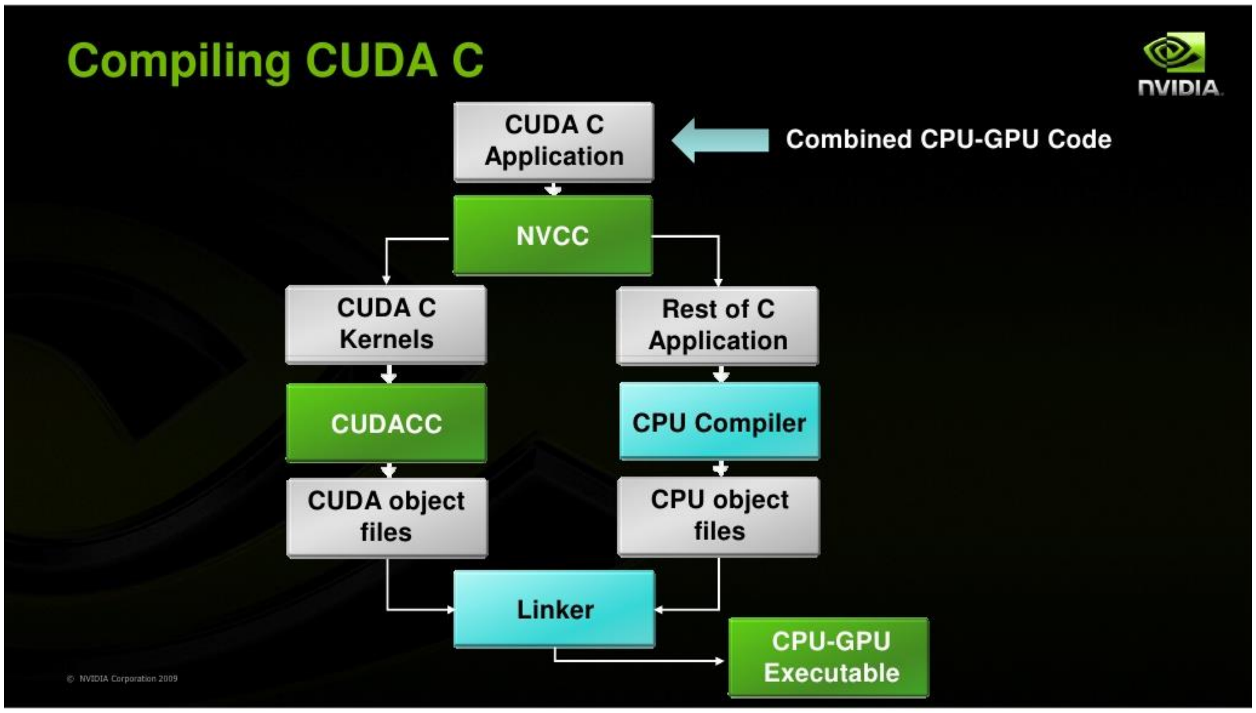
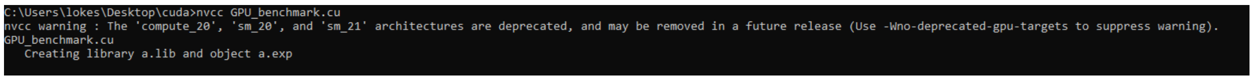


Fig.7 Compiler

RESULTS

1. Compilation of the program

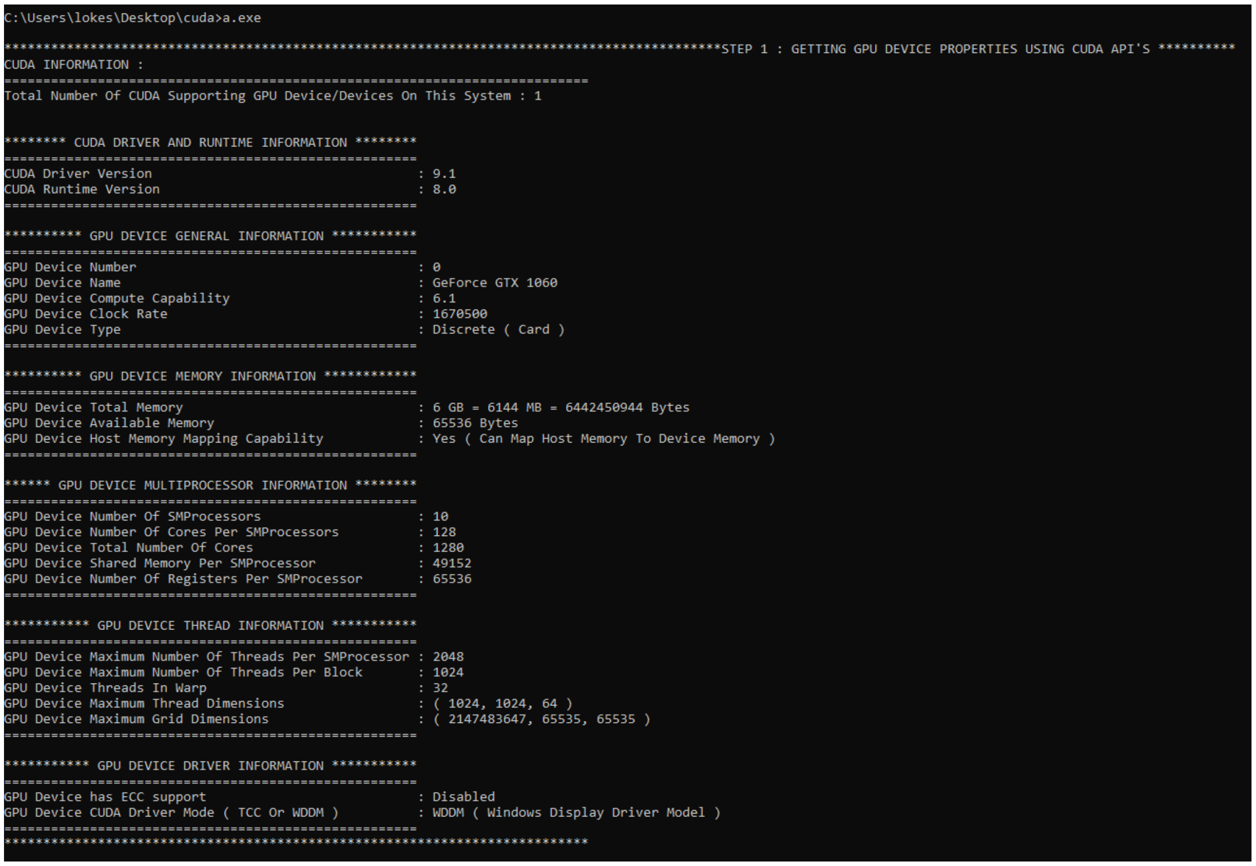


```
C:\Users\lokes\Desktop\cuda>nvcc GPU_benchmark.cu
nvcc warning : The 'compute_20', 'sm_20', and 'sm_21' architectures are deprecated, and may be removed in a future release (Use -Wno-deprecated-gpu-targets to suppress warning).
GPU_benchmark.cu
   Creating library a.lib and object a.exp
```

“a.exe” executable file is generated after compilation process.

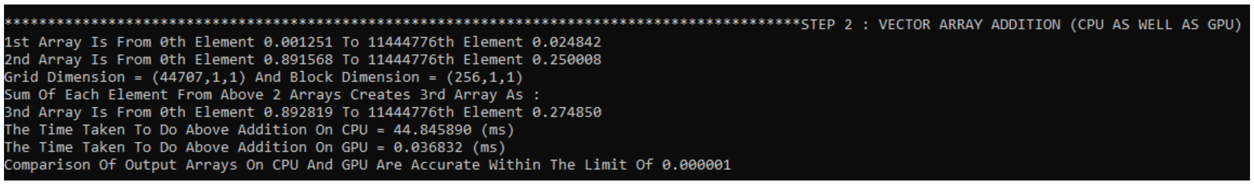
Execution of code is as below:

STEP 1: Getting GPU device properties using CUDA API



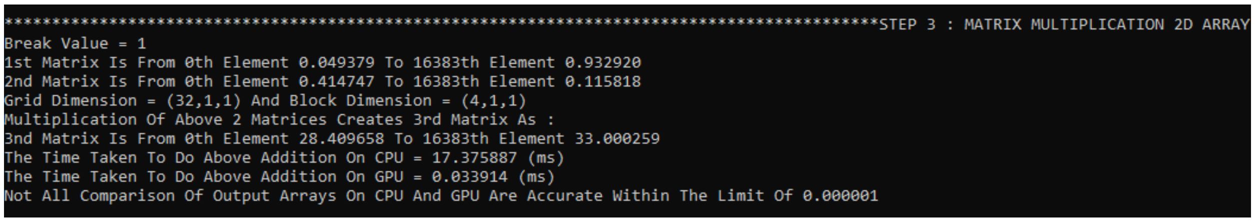
```
C:\Users\lokes\Desktop\cuda>a.exe

********************************************************************************************STEP 1 : GETTING GPU DEVICE PROPERTIES USING CUDA API'S **********
CUDA INFORMATION :
===========================================================================
Total Number Of CUDA Supporting GPU Device/Devices On This System : 1


******** CUDA DRIVER AND RUNTIME INFORMATION ********
=====================================================
CUDA Driver Version                                  : 9.1
CUDA Runtime Version                                 : 8.0
=====================================================

********** GPU DEVICE GENERAL INFORMATION ***********
=====================================================
GPU Device Number                                    : 0
GPU Device Name                                      : GeForce GTX 1060
GPU Device Compute Capability                        : 6.1
GPU Device Clock Rate                                : 1670500
GPU Device Type                                      : Discrete ( Card )
=====================================================

********** GPU DEVICE MEMORY INFORMATION ************
=====================================================
GPU Device Total Memory                              : 6 GB = 6144 MB = 6442450944 Bytes
GPU Device Available Memory                          : 65536 Bytes
GPU Device Host Memory Mapping Capability            : Yes ( Can Map Host Memory To Device Memory )
=====================================================

****** GPU DEVICE MULTIPROCESSOR INFORMATION ********
=====================================================
GPU Device Number Of SMProcessors                    : 10
GPU Device Number Of Cores Per SMProcessors          : 128
GPU Device Total Number Of Cores                     : 1280
GPU Device Shared Memory Per SMProcessor             : 49152
GPU Device Number Of Registers Per SMProcessor       : 65536
=====================================================

*********** GPU DEVICE THREAD INFORMATION ***********
=====================================================
GPU Device Maximum Number Of Threads Per SMProcessor : 2048
GPU Device Maximum Number Of Threads Per Block       : 1024
GPU Device Threads In Warp                           : 32
GPU Device Maximum Thread Dimensions                 : ( 1024, 1024, 64 )
GPU Device Maximum Grid Dimensions                   : ( 2147483647, 65535, 65535 )
=====================================================

*********** GPU DEVICE DRIVER INFORMATION ***********
=====================================================
GPU Device has ECC support                           : Disabled
GPU Device CUDA Driver Mode ( TCC Or WDDM )          : WDDM ( Windows Display Driver Model )
=====================================================
***************************************************************************
```

STEP 2: Addition of vector array



```
********************************************************************************************STEP 2 : VECTOR ARRAY ADDITION (CPU AS WELL AS GPU)
1st Array Is From 0th Element 0.001251 To 11444776th Element 0.024842
2nd Array Is From 0th Element 0.891568 To 11444776th Element 0.250008
Grid Dimension = (44707,1,1) And Block Dimension = (256,1,1)
Sum Of Each Element From Above 2 Arrays Creates 3rd Array As :
3nd Array Is From 0th Element 0.892819 To 11444776th Element 0.274850
The Time Taken To Do Above Addition On CPU = 44.845890 (ms)
The Time Taken To Do Above Addition On GPU = 0.036832 (ms)
Comparison Of Output Arrays On CPU And GPU Are Accurate Within The Limit Of 0.000001
```

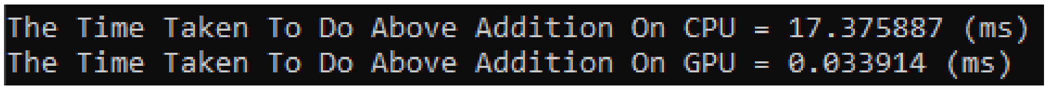
STEP 3: Multiplication of the 2D array



```
********************************************************************************************STEP 3 : MATRIX MULTIPLICATION 2D ARRAY
Break Value = 1
1st Matrix Is From 0th Element 0.049379 To 16383th Element 0.932920
2nd Matrix Is From 0th Element 0.414747 To 16383th Element 0.115818
Grid Dimension = (32,1,1) And Block Dimension = (4,1,1)
Multiplication Of Above 2 Matrices Creates 3rd Matrix As :
3nd Matrix Is From 0th Element 28.409658 To 16383th Element 33.000259
The Time Taken To Do Above Addition On CPU = 17.375887 (ms)
The Time Taken To Do Above Addition On GPU = 0.033914 (ms)
Not All Comparison Of Output Arrays On CPU And GPU Are Accurate Within The Limit Of 0.000001
```

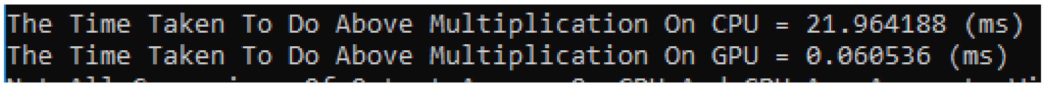
Time taken by GPU and CPU for different code snippets are given below,

Vector addition of 2 arrays (Array size 11444777 values)



```
The Time Taken To Do Above Addition On CPU = 17.375887 (ms)
The Time Taken To Do Above Addition On GPU = 0.033914 (ms)
```

2D array multiplication of two arrays(Each array of 128 rows, 128 columns)



```
The Time Taken To Do Above Multiplication On CPU = 21.964188 (ms)
The Time Taken To Do Above Multiplication On GPU = 0.060536 (ms)
```

FINDINGS/CONCLUSIONS

Performance of CPU Vs multiple GPU's is evaluated, and time taken to execute is recorded. In this project, addition of two huge array of numbers (approx. 11million values) and multiplication two 2D array each has 128 rows and 128 columns was given as input and computed on CPU(Intel i7 processor) as well as on GPU and the results were recorded, also tested the time required to process it. Also, benchmark program is tested on different computers which has Nvidia GPU to compare the results.

REFERENCES

[1] https://en.wikipedia.org/wiki/Graphics_processing_unit.

[2] https://en.wikipedia.org/wiki/CUDA.

[3] www.nvidia.com/docs/IO/116711/sc11-cuda-c-basics.pdf.

[4] https://web.stevens.edu/sss/summer-2010/7-7-2010-CUDA_Programming.

[5] www.cs.kent.edu/~wcheng/GA%20works.

[6] www.glearning.tju.edu.cn.

[7] Computer Architecture In-Class Slides.