FINAL PROJECT REPORT: CDA5155 COMPUTER ARCHITECTURE PRINCIPLES

Title: GPU Computing Evolution and Benchmarking

Team members: Chien Chi Lee, Nikhil Tank

PROJECT OUTLINE:

1. Abstract
2. Introduction
3. GPU computing's evolution
    a. Early GPUs
    b. GPU technology development (till 2010)
    c. GPU development in the next decade
4. Application performance (Experimentation)
    a. Speed up for High-Quality Image Contour Detection using Nvidia GeForce MX130 (Mobile GPU - 2018)
    b. Performance Comparison using market benchmarking software
    c. Performance comparison over array addition over a conventional CPU
5. References

Title: GPU Computing Evolution and Benchmarking
Team members: Chien Chi Lee, Nikhil Tank

1. Abstract:
As the emergence of the graphics processing unit (GPU), GPUs has become an integral part of mainstream computing system nowadays. The modern GPU is not only a powerful graphics engine but also a highly-parallel programmable processor that substantially outpaces its CPU counterpart. GPUs are used in various demanding consumer applications. They have high-performance computing on manipulating image processing and computer graphics. This report will be focused on the rapid evolution of GPU architecture on each generation from the early to modern GPU architecture, and also evaluate the performance with respect to the traditional CPUs.

2. Introduction:
As the demand of parallel computing with high performance increases across many areas of science, medicine, engineering and finance, the chip manufacturers continue to innovate and meet the demand with extraordinary powerful GPU. The High-Performance Computing (HPC) domains like seismic processing, biochemistry solutions, weather and climate modelling, signal processing, computational finance, and data analysis. GPUs are designed to help solve the world/s most difficult computing problems.

3. GPU computing's evolution
GPU computing provides a huge advantage over the CPU with respect to computing speed. Thus, GPU computing has become one of the most popular areas of research in the field of modern development and industrial research. This article first describes the GPU from the early stage with a configurable graphics processor to a programmable parallel processor.
In order to introduce the evolution of GPU computing, this article uses NVIDIA GPUs as examples with several architectures including Fermi Microarchitecture, Kepler Architecture, and Turing Microarchitecture.

a. Early GPUs
In the 1990s there were no GPUs in this generation. The way to accelerate graphical graphical user interfaces (GUI) was to use 2D graphics displays for PCs which was generated by video graphics array (VGA). During the 1990s, integration of graphics chips grew up because the manufacturing capabilities were better and improved than before. Therefore, 2D GUI acceleration was evolved continuously. In the mid-1990s, the increasing demand in public for accelerated 3D graphics was due to real time 3D graphics used commonly in computer, arcade, and games areas. NVIDIA released the RIVA 3D single chip graphics to accelerate games 3D and visualization applications in 1997. The chip graphics accelerator was programmed with Microsoft Direct3D and OpenGL. It led evolution for modern GPUs from fixed function pipelines to microcode, configurable, and scalable parallel processors [1].

In 1994, the term "GPU" was coined by Sony to refer the PlayStation console Sony GPU. The term was popularized by Nvidia in 1999, marked the GeForce 256 as "the world's first GPU"." The first GPU which was the GeForce 256 was a single-chip 3D real time graphics processor. GeForce 256 included many features of high-end workstation graphics pipelines. 32-bit floating-point vertex transform and lighting processor, and integer pixel-fragment pipeline were contained in GeForce 256. GPUs calculated 3D geometry and vertices by first using floating-point arithmetic. GPUs were applied to pixel lighting and Color values to simplify programming and also to handle high-dynamic-range scenes.

b. GPU Technology development

In 2001, NVIDIA released the GeForce 3 which introduced the first programmable vertex processor that executed vertex shader programs. It was also along with 32-bit floating-point pixel fragment pipeline and was programmed with OpenGL and DX8. In 2002, ATI Radeon 9700 introduced a programmable 24-bit floating-point pixel-fragment processor which was programmed with OpenGL and DX9. The GeForce 6800 and GeForce FX [5] featured vertex processors, programmed with Cg programs, DX9, and OpenGL, and programmable 32-bit floating-point pixel fragment processors. The processors mentioned above were multithreading that can create a thread to execute a thread program for each pixel fragment and vertex. The GeForce 6800 facilitated multiple GPU implementations with different numbers of processor cores because of its scalable processor core architecture.

From 2006 to 2009, Nvidia released the GeForce 8 series and make new generic stream processing unit GPUs become more generalized computing devices. Nowadays, parallel GPUs have made computational inroads against the CPU. In 2006, the GeForce 8800 which was introduced by Nvidia featured the first unified graphics and computing GPU architecture [2][3] programmable in C programming language with the CUDA parallel computing system instead of using OpenGL and DX. The GeForce 8800's unified streaming processor cores could execute computing threads for CUDA C programs. and also execute vertex, pixel shader threads, and geometry for DX10 graphics programs. The GeForce 8800 efficiently executes most up to 12288 threads in 128 processor cores concurrently by hardware multithreading. Besides, the GeForce 8800 as the first GPU used scalar thread processors to match standard scalar languages like C language, and also to eliminate the demand to program vector operations and manage vector registers. The GeForce 8800 have added instructions to support the C language and other general-purpose languages, which includes load/store memory access instructions with byte addressing integer arithmetic, and floating-point arithmetic. It also provided instructions and hardware to support parallel computation, synchronization, and communication.

CUDA architecture

Compute Unified Device Architecture (CUDA) created by NVIDIA featured in the GPU cards for general purpose computing with GPUs. CUDA has the advantage of massive computational power on programming, provided by Nvidia graphics cards [4]. CUDA provides 128 processors cores concurrently to run multithreaded applications. CUDA is useful for highly parallel algorithms. In general, if there are more numbers of threads, the performance will be much better. However, CUDA is not useful for most of the serial algorithms. If the problem mentioned above cannot be broken into at least one thousand threads, using CUDA has no huge advantage. GPUs have large amount of arithmetic capability to increase programmability in the pipeline.

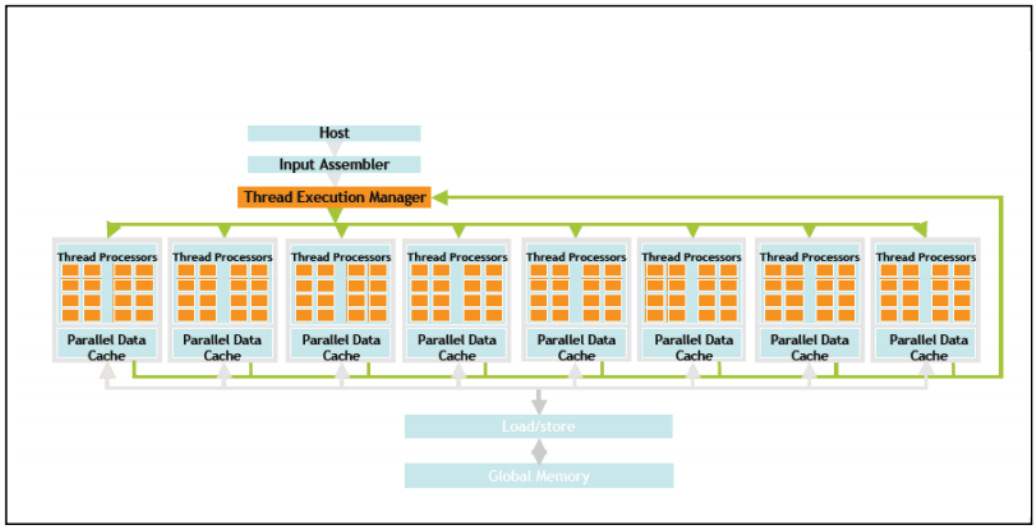

Figure1: GPU Architecture [5]

The architecture of a typical CUDA-capable GPU is shown in Fig. 1. CUDA is capable of high degree of threading which is an array of streaming processors. In Fig. 1, two Streaming Multiprocessor (SM)

is shown to form a building block. However, the number of SM in a building block depends on the generation of CUDA GPUs. Each GPU has a global memory which comes with 4 gigabytes of graphics double data rate DRAM. For graphics applications, they hold texture information for 3D rendering, and video images. However, their function for computing as off-chip memory, very-high-bandwidth, causing more latency than typical system memory.

c. GPU development in the next decade

During 2010 Nvidia released the GeForce GTX 470 and GTX 480, these were the first cards with the Fermi architecture. But they got heavily criticized because of their high-power consumption, along with its inefficiency to dissipate heat properly. The predecessor was replaced by the more balanced GTX580 which had the issues resolved.

Fermi Microarchitecture (2010) [19]

The Fermi GPU architecture consisted of Multiple Streaming multiprocessors (SMs), each had 32 cores, each capable of doing 1 floating point or integer instruction per clock. The SMs were supported by a second-level cache, host interface, GigaThread scheduler and multiple DRAM interfaces. Fermi was the oldest microarchitecture from NVIDIA that received support for the Microsoft's rendering API Direct3D.

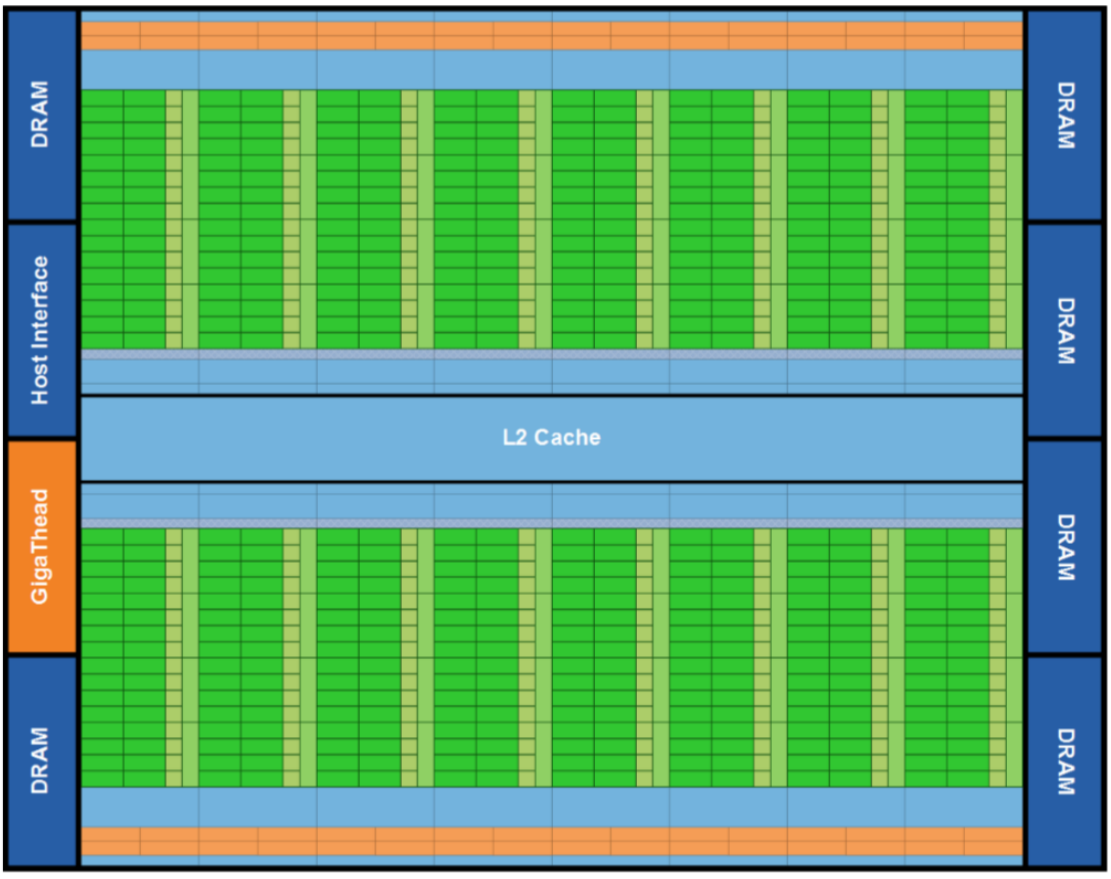

Figure2: Fermi Architecture [19]

Each SM had 32 CUDA processors, it was 4 times increment form the prior Tesla SM design. The Fermi architecture used the IEEE 754-2008 floating pt. standard, which provided the new concept of Fused multiply-add (FMA) instruction for both single and double float pt. operation.

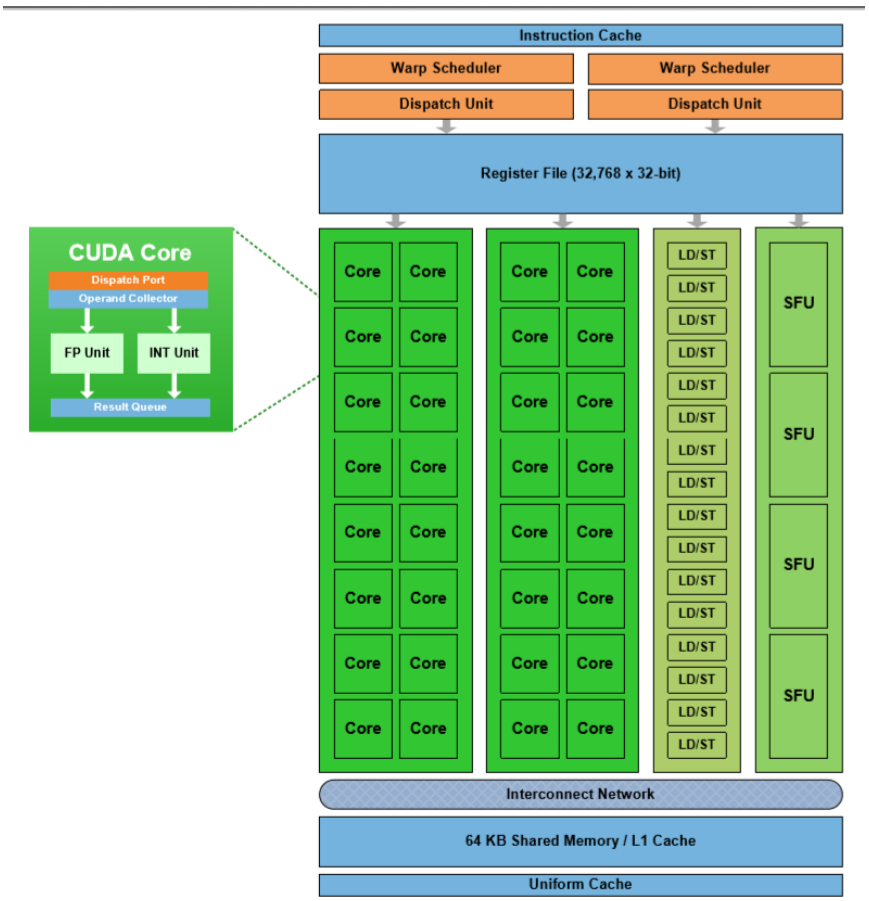

Figure 3: Fermi Architecture SM [19]

Fused multiply-add: It provided a feature of doing the multiply and add with a single rounding step, with no loss of precision during the addition. [19]

Kepler Architecture (2012) [20]

The Kepler architecture incorporated many new innovative features that enabled increased GPU utilization, simplified parallel program design. The Dynamic parallelism, Hyper Q, Grid Management Unit were first time introduced in Kepler.

Dynamic Parallelism added the capability for the GPU to assign or generate new work for itself when ideal, and control the scheduling of the work through dedicated hardware paths, without even involving CPU. It provided the flexibility to adapt to the amount and form of parallelism through the course of a program's execution. This capability allows less-structured, more complex tasks to run easily and effectively, enabling large percentage of application running on GPU entirely. Resulting in freed CPU for other tasks.

Hyper Q enables multiple CPU cores to launch work on a single GPU simultaneously, therefore reducing the CPUs consumption and increase utilized GPU. The Hyper-Q is a flexible solution that allows separate connections from the multiple CUDA streams, from multiple Message Passing Interface (MPI) processes, or even from multiple threads within a process, this dramatically increase the performance without changing existing code. [20] The successor of Kepler was Maxwell Microarchitecture (2013), which introduced an improved Streaming Multiprocessor (SM) design that increased power efficiency.

Pascal Microarchitecture (2016) [22]

This generation used the 16nm FinFET process and boost clock of 1733. Along side of the fact that Pascal Architecture was a die-shrink of the Maxwell Architecture, but also introduces numerous optimization and new features like GPU Boost 3.0 allowing overclocking. High-performance computing applications are bottlenecked by memory bandwidth. Developers put a lot of efforts to optimizing code for efficient memory accesses, and to keep data in the parts of the memory hierarchy closest to the computational units. Some applications—for example in deep learning where many-layered neural networks are trained using massive data sets—are limited more by memory capacity. Hence memory poses two challenges: bandwidth and capacity. Tesla P100 accelerator battles both issues using stacked memory, enabling multiple layers of DRAM components to be integrated vertically on the package along with the GPU. Tesla P100 is the first GPU accelerator to use High Bandwidth Memory 2 (HBM2). It provides more than twice the capacity, and higher energy efficiency, compared to current off-package GDDR5.

Figure4: Pascal Architecture [22]

Architecture	***No. of CUDA cores per SM***
Pascal	128
Maxwell	128
Kepler	192
Fermi	32
Tesla	32

GPU / FORM FACTOR [21]	**KEPLER GK110 / PCIE**	**MAXWELL GM200 / PCIE**	**PASCAL GP100 / SXM2**	**PASCAL GP100 / PCIE**
TESLA PRODUCTS	**Tesla K40**	**Tesla M40**	**Tesla P100 (NVLink)**	**Tesla P100 (PCIe)**
SMS	15	24	56	56
FP64 CUDA CORES / SM	64	4	32	32
FP64 CUDA CORES / GPU	960	96	1792	1792
BASE CLOCK	745 MHz	948 MHz	1328 MHz	1126 MHz
GPU BOOST CLOCK	810/875 MHz	1114 MHz	1480 MHz	1303 MHz
MEMORY BANDWIDTH	288 GB/s	288 GB/s	732 GB/s	549 GB/s (12GB) 732 GB/s (16GB)
MEMORY SIZE	Up to 12 GB	Up to 24 GB	16 GB	12 GB or 16 GB
L2 CACHE SIZE	1536 KB	3072 KB	4096 KB	4096 KB
TRANSISTORS	7.1 billion	8 billion	15.3 billion	15.3 billion
MANUFACTURING PROCESS	28-nm	28-nm	16-nm	16-nm

Architecture Comparison - Data courtesy [23]

Turing Microarchitecture (2018) [21]

This is the latest architecture by Nvidia, which is claimed to represent the biggest architecture leap forward in over a decade. Turing GPUs introduce new RT Cores, accelerator units that are dedicated to performing ray tracing operations with great efficiency. It has two key architectural changes:

- It adds a new independent integer Datapath that can execute instructions concurrently with the floating pt. math Datapath. In old generations these executions would have blocked floating pt. instructions from issuing.
- The SM memory path has been redesigned to unify shared memory, and load caching into one unit. This provides 2X more bandwidth and more than 2X more capacity available for L1 cache. This enables to achieve 50% improvement in delivering performance per CUDA code.

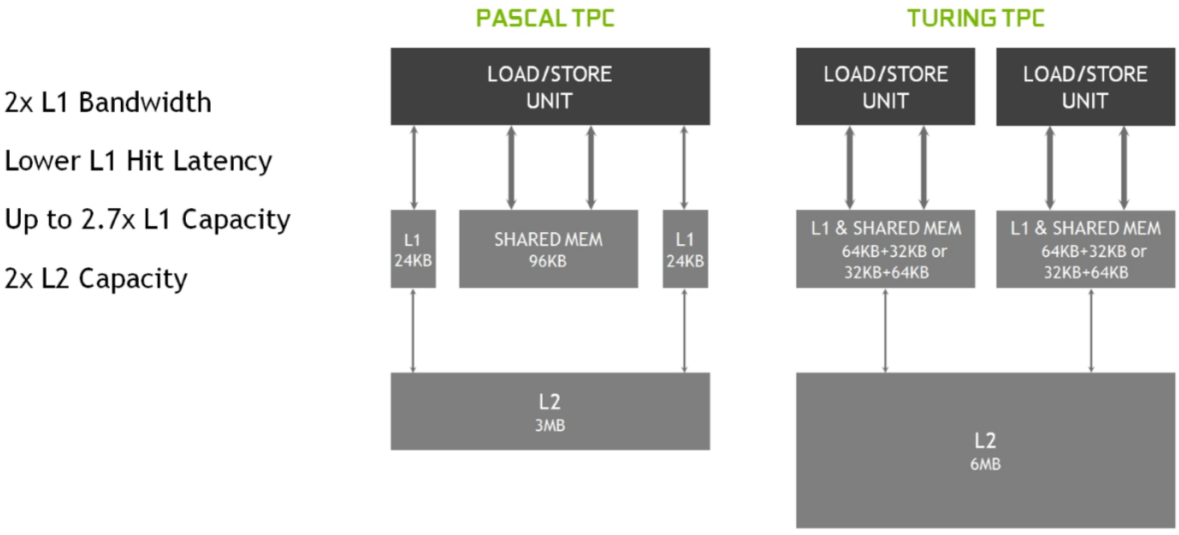

Figure5: Shared memory architecture [21]

Turing GPUs includes an updated version of the Tensor Core. It has "576 Tensor cores: eight per SM and two per each processing block within an SM. Each capable of performing up to 64 floating pt. fused multiply-add (FMA) operations per clock using F16 inputs." [21] It performs 1024 total FP operation per clock. In Pascal, for example, every time an integer instruction was run the dual-purpose CUDA core couldn't do anything else - it was either floats or integers. The Truing GPU has a new approach for Ray tracing, which is a computationally-intensive rendering technology that realistically simulates the lighting of a scene and the objects. Turing GPU's RT technology can render physically correct reflection in real time which was earlier not possible with a single GPU.

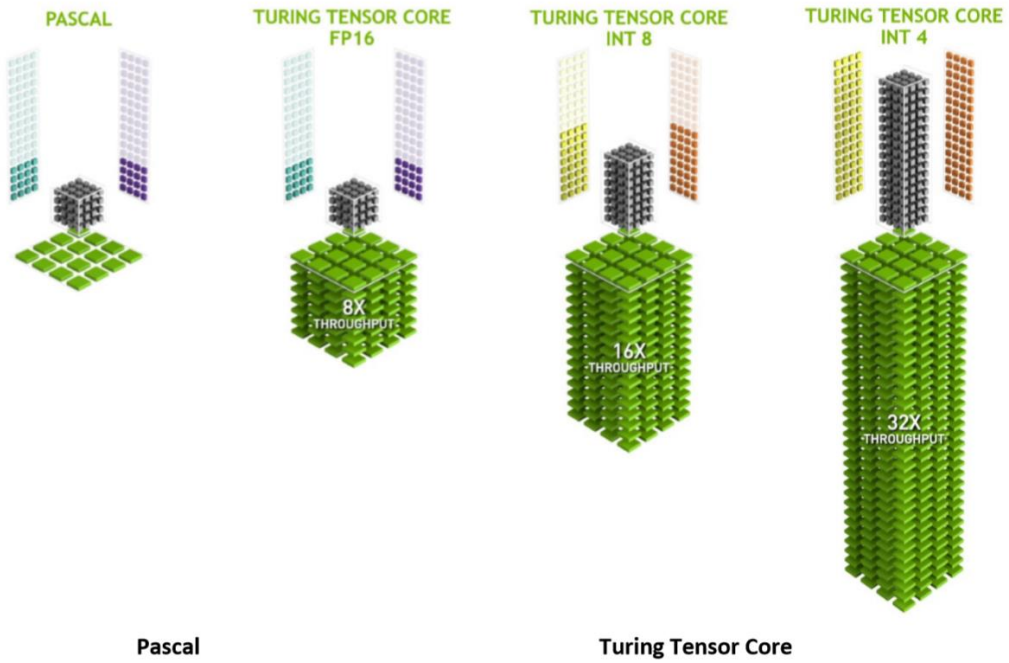

Figure6: Comparison of Turing Tensor Core with Pascal [21]

Mobile GPUs

The mobile GPU is a dedicated co-processor designed to accelerate, user interfaces, 3D content and graphics applications on your IoT devices, tablet, and smartphone. live graphical user interfaces (GUIs) and Photorealistic 3D games are examples of designing specifically for the GPU. The GPU has become an indispensable part of product specification for all mobile application processor with graphical displays being everywhere and widely used on all kinds of connected devices. Also, the GPU makes product differentiation so companies can create individual product on visual-centric solutions, and compelling for their target application[8].

While performance is the main goal for desktop GPUs, mobile GPUs must balance performance against power consumers. To reduce the enormous bandwidth demand, one of the biggest consumers of power in a device, many mobile GPUs use tile-based rendering (TBR) approach. TBR is to break up the OpenGL framebuffer similar tiles and render once during a time. For each tile, all the primitives that affect it are rendered into tile buffer, and once the tile is complete, it is copied back to the more power-hungry main memory. The bandwidth advantage is due to only having to write back a minimum set of results. Additionally, blending, depth testing, and stencil testing are entirely done on-chip.

4. Application performance (Experimentation)

a. Speed up for Image processing using Nvidia GeForce MX130 (Mobile GPU - 2018)

GPU proves their usefulness in image processing applications. Operations like Filtering require operations like addition, multiplication, convolution, FFTs and IFFTs over every pixel of the image, and GPU proves its expertise in these domains.

Taking an example of Bilateral filter, which is an edge-preserving non-linear smoothing filter that is implemented with CUDA with OpenGL rendering. It is used in image recovery and denoising domain. Each pixel is weight by considering both the spatial distance and Color distance between its neighbours. [10]

Using the system configuration as Intel Core i5, 8th Gen and MX130 GPU the Bilateral filter was first solved using on the CPU. Implementation of bilateral filter on CPU is computationally intensive as it requires calculation of neighbour's weight at each pixel of image over a weight kernel. So, the image size was kept 480 X 480 pixels. The CPU code was written in Python and no library was used. Running the code for an image of Century tower took 346.36 secs.

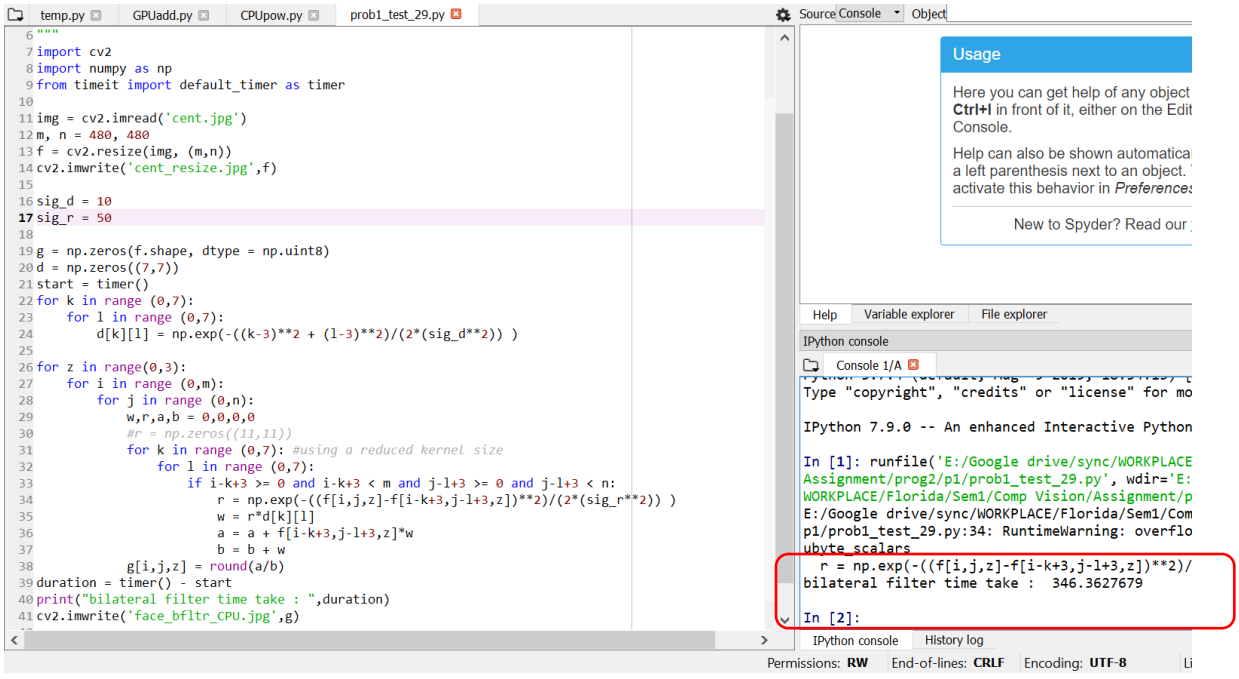

Figure7: CPU execution of Bilateral Filter

Figure8: Century tower Image

Figure9: Bilateral Century tower

The Nvidia CUDA Toolkit 10.2 [16] comes with pre built CUDA samples code which can be implemented using the Visual Basic C++. Bilateral Filtering is filter code is also a part of this. The implementation of this is displayed in the Figure 10. The GPU turned out to be so effective in this task that for the image size of 480 X 640 pixels the operation was real time with changing sigma variable. From the Visual Basic's diagnostics session, it turns out that 145 MB memory and 14 % of all process's CPU consumption was recorded.

bilateral_kernel.cu | bilateralFilter_cpu.cpp | bilateralFilter.cpp

bilateralFilter | (Global Scope)

```
84
85      //column pass using coalesced global memory reads
86      __global__ void
87      d_bilateral_filter(uint *od, int w, int h,
88                         float e_d,  int r, cudaTextureObject_t rgbaTex)
89      {
90          int x = blockIdx.x*blockDim.x + threadIdx.x;
91          int y = blockIdx.y*blockDim.y + threadIdx.y;
92
93          if (x >= w || y >= h)
94          {
95              return;
96          }
97
98          float sum = 0.0f;
99          float factor;
100         float4 t = {0.f, 0.f, 0.f, 0.f};
101         float4 center = tex2D<float4>(rgbaTex, x, y);
102
103         for (int i = -r; i <= r; i++)
104         {
105             for (int j = -r; j <= r; j++)
106             {
107                 float4 curPix = tex2D<float4>(rgbaTex, x + j, y + i);
108                 factor = cGaussian[i + r] * cGaussian[j + r] *     //domain factor
109                          euclideanLen(curPix, center, e_d);            //range factor
110
111                 t += factor * curPix;
112                 sum += factor;
113             }
114         }
115
116         od[y * w + x] = rgbaFloatToInt(t/sum);
117     }
```

82 % | 10 | 0

Figure10: Bilateral Filter Execution on GPU C++ CUDA kernel

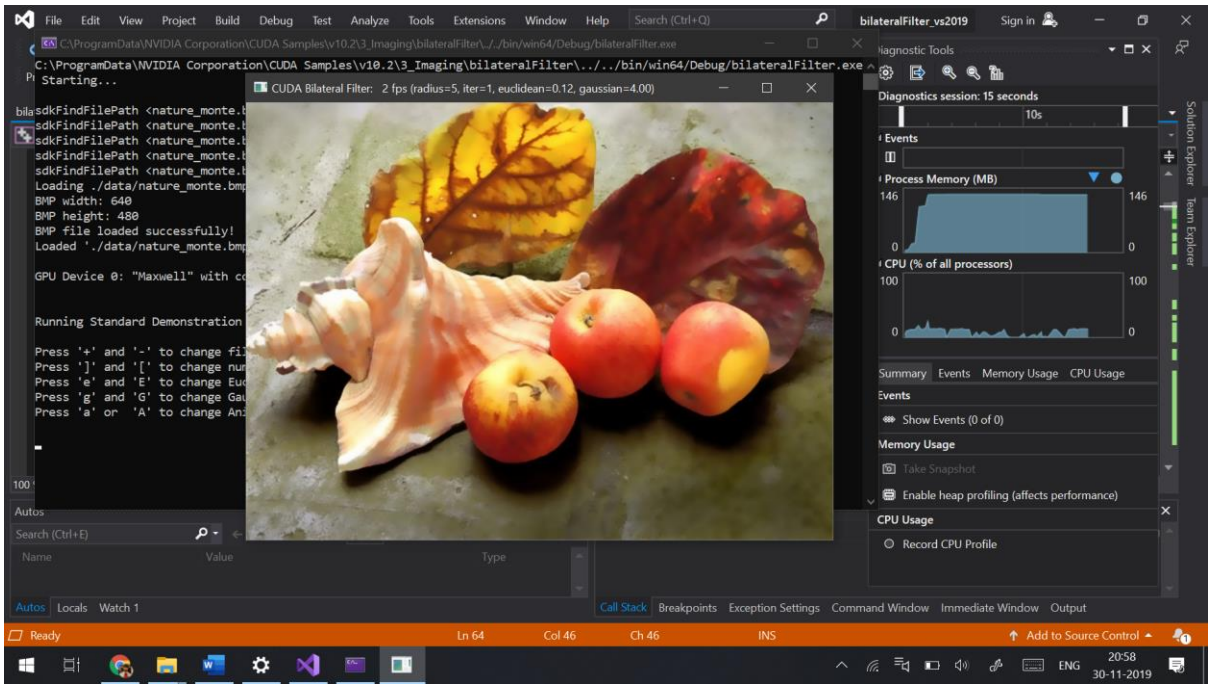

Figure11: Bilateral Filter Execution on GPU output (C++ CUDA code)

b. Performance Comparison using market benchmarking software:

The Clock cycle time is the quanta in microprocessor industry. It is the amount of time between 2 pulses of an oscillation. The Graphic acceleration software tweak this by changing clock cycle duration. Overclocking increases the component clock rate, and make the component run at a higher speed than it was designed to run. The overclocking is a tricky as it works on the fine line between better performance and broken hardware. The application such as “MSI” recognizes this and has created its own overclocking tool within Afterburner [11] this simplifies the process. The overclocking feature provides precision settings that allow user to push the GPU to its maximum potential while protecting the hardware from permanent damage caused by insufficient heat decapitation. This software can tweak almost every aspect of a GPU, including clock speeds, core voltage, power limit, temperature limit, memory clock speed, and fan speed. Nvidia also released GPU Boost 2.0 (2010), which would allow the GPU clock speed to increase indefinitely until a user-set temperature limit was reached without passing a user-specified maximum fan speed.

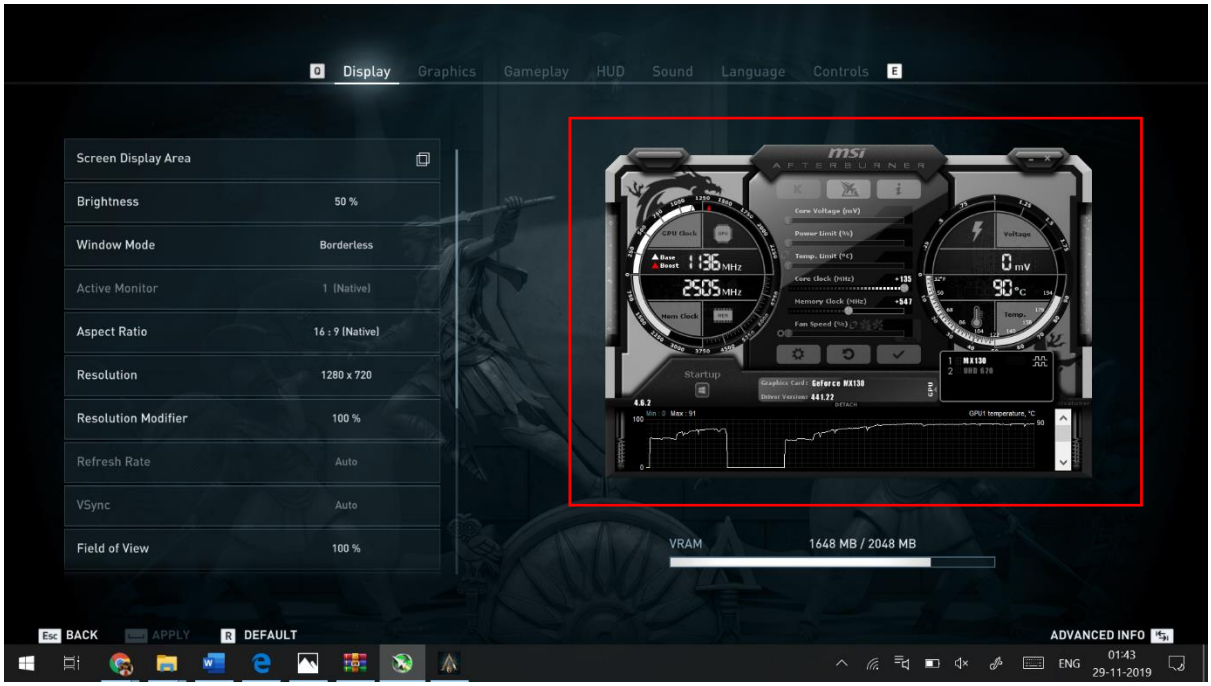

Figure12: MSI Afterburn User Interface [11]

Afterburner uses OC Scanner, a utility that uses an algorithm developed by Nvidia to determine any model of GPU (whether it's a 960 or 2080). Once the OC Scanner has determined what card the user is running, it will work out the highest and most stable overclocking settings for that specific card. Then, it applies the changes to your card for instantaneous performance gains. This can be seen in games with better achieved graphic quality or better Frames Per Second (FPS).

The performance graph for while playing the game Ubisoft's Assassins Creed Odyssey is calculated on the MX130 GPU with and without MSI Afterburn. From the Fig it is clear that with MSI Afterburn we have achieved 30 FPS (+4) in place of 26 FPS.

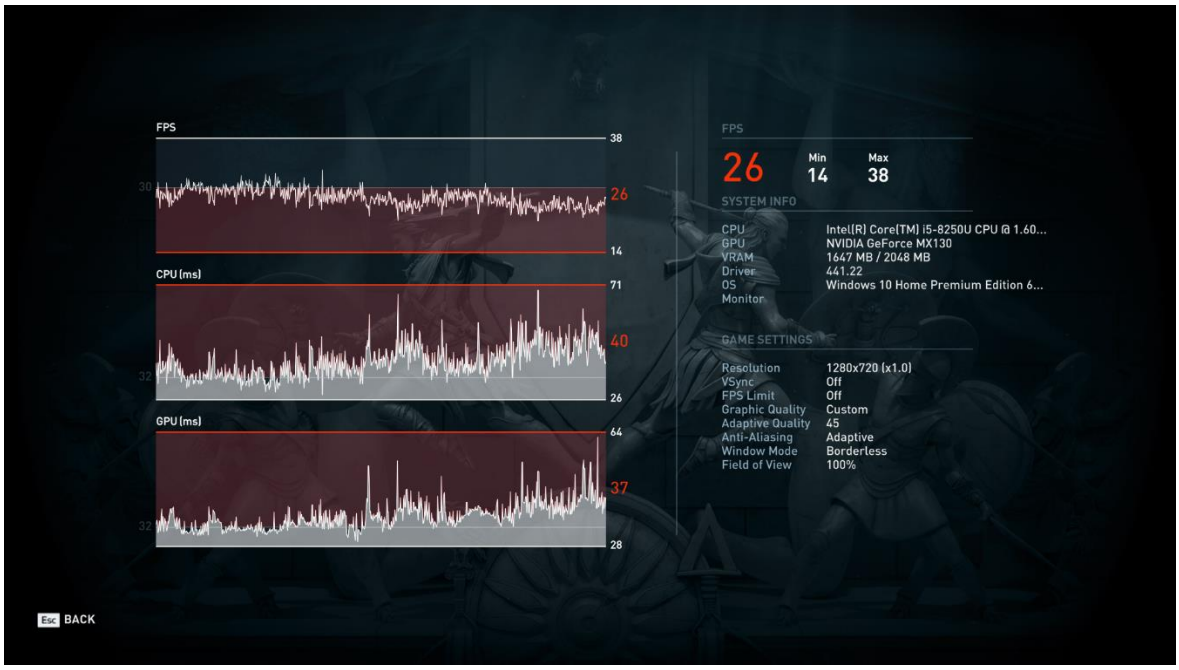

Figure13: Without over clocking - Ubisoft's Assassins Creed Odyssey

Figure14: With overclocking with +135 MHz, MSI Afterburn

Benchmarking: Surrounded with competition in GPU market, benchmarking has become very popular amongst product testers nowadays. It is evident that people want to know the performance of different GPUs card and this require a fair, level playing field, benchmarking test.

MSI had developed a different benchmarking tool called *MSI Kombustor* [12]. It runs a serious of stressful GPU processes to see how well your graphics card reacts to the pressure. It's a great way to see whether or not your overclocking tweaks have made an impact on the performance of your GPU.

Kombustor 4.1.1.0 - score ID: 111337

MSI-01 (TESS+PBR+DOF)

Submitted by anonymous on Nov 29, 2019 @ 07:55:10

SCORE	166
FPS	3
3D Renderer	Intel(R) UHD Graphics 620
3D API	OpenGL 4.5.0 - Build 25.20.100.6446
Resolution	1920x1080
Duration	60000
CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
OS	Windows 10 build 18362
GPU 0	GeForce MX130

Figure15: Benchmarking score without overclock

Kombustor 4.1.1.0 - score ID: 111341

MSI-01 (TESS+PBR+DOF)

Submitted by anonymous on Nov 29, 2019 @ 07:57:57

SCORE	171
FPS	3
3D Renderer	Intel(R) UHD Graphics 620
3D API	OpenGL 4.5.0 - Build 25.20.100.6446
Resolution	1920x1080
Duration	60000
CPU	Intel(R) Core(TM) i5-8250U CPU @ 1.60GHz
OS	Windows 10 build 18362
GPU 0	GeForce MX130
GPU 0 device ID	10DE-174D
Driver	R441.22
GPU 1	Intel(R) UHD Graphics 620
GPU 1 device ID	8086-5917

page generated in 0.0006020069122314 5 seconds

Figure16: Benchmarking score with overclock

c. Performance comparison over array addition over a conventional CPU

GPUs are designed with an ability of performing repetitive task in less time. Problem like vector and matrix operation are performed rapidly using the multiple CUDA core design. Taking the vector addition problem, the CPU and GPU codes were written and the matrix sizes were taken considerably big as **100000000 elements**. The evaluation parameter is taken her to be the run time. Using Intel Core i5, 8th Gen CPU the program took **41.58 sec** while when the code was run on MX 130 GPU the program took **1.85 sec**. It should be noted that the run time is not an absolute measure for performance but gives a sense of how efficiently the parallel operation is achieved using the GPU architecture. This code is written in Python and uses the @Vectorize function in python from Numba library.

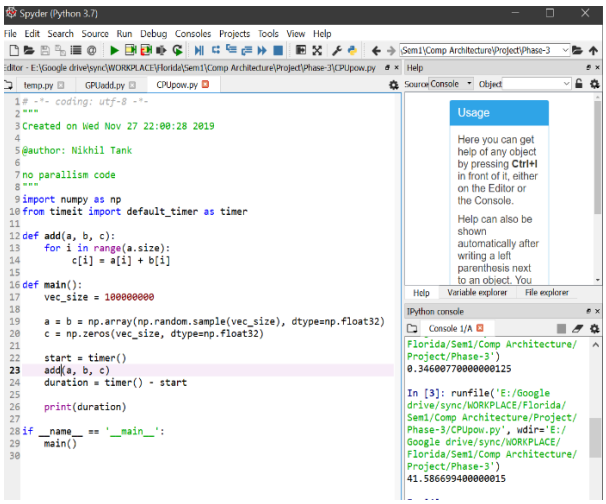

Figure17: CPU code

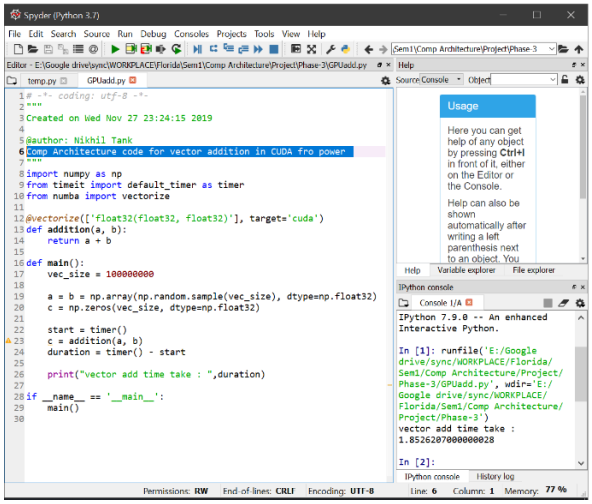

Figure18: GPU code

```
5 @author: Nikhil Tank
6 Comp Architecture code for vector addition in CUDA fro power
7 """
8 import numpy as np
9 from timeit import default_timer as timer
10 from numba import vectorize
11
12 @vectorize(['float32(float32, float32)'], target='cuda')
13 def addition(a, b):
14     return a + b
15
16 def main():
17     vec_size = 100000000
18
19     a = b = np.array(np.random.sample(vec_size), dtype=np.float32)
20     c = np.zeros(vec_size, dtype=np.float32)
21
22     start = timer()
23     c = addition(a, b)
24     duration = timer() - start
25
```

Figure19: CUDA Kernel for Vector Add (Python)

Matrix (320,320) Matrix (640,320)	**Computing result using CUDA Kernel...**
Performance	2.10 GFlop/s,
Time	62.411 msec,
Size	131072000 Ops
Workgroup Size	1024 threads/block

Figure20: CUDA output for Matrix Multiplication (C++)

5. Reference:

[1] J. Montrym and H. Moreton, ‘‘The GeForce 6800,’’ IEEE Micro, vol. 25, no. 2, 2005, pp. 41-51

[2] E. Lindholm et al., ‘‘NVIDIA Tesla: A Unified Graphics and Computing Architecture,’’ IEEE Micro, vol. 28, no. 2, 2008, pp. 39-55.

[3] J. Nickolls and D. Kirk, ‘‘Graphics and Computing GPUs,’’ Computer Organization and Design: The Hardware/Software Interface, D.A. Patterson and J.L. Hennessy, 4th ed., Morgan Kaufmann, 2009, pp. A2-A77.

[4] Ghorpade, Jayshree. (2012). GPGPU Processing in CUDA Architecture. Advanced Computing: An International Journal. 3. 105-120. 10.5121/acij.2012.3109.

[5] Anthony Lippert – “NVidia GPU Architecture for General Purpose Computing”

[6] https://developer.nvidia.com/pascal

[7] https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf

[8] https://www.embedded-computing.com/embedded-computing-design/understand-the-mobile-graphics-processing-unit?fbclid=IwAR13DWj4O7zba0ekJHi8JKya5KceMPr9U8Rcd4tfUL_6NQDbNDiYrsDTZIM

[9] https://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_Architecture.pdf

[10] C. Tomasi, R. Manduchi, Bilateral Filtering for Gray and Color Images, proceeding of the ICCV, 1998, http://users.soe.ucsc.edu/~manduchi/Papers/ICCV98.pdf

[11] https://www.msi.com/page/afterburner

[12] https://geeks3d.com/furmark/kombustor/downloads/

[13] https://docs.nvidia.com/cuda/cuda-installation-guide-microsoft-windows/index.html

[14] https://developer.nvidia.com/cuda-downloads

[15] https://www.cs.utah.edu/~mhall/cs4961f10/CUDA-VS08.pdf

[16] https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html

[17] https://www.nvidia.com/content/DriverDownload-March2009/confirmation.php?url=/Windows/441.41/441.41-notebook-win10-64bit-international-whql.exe&lang=us&type=GeForcem

[18] https://www.youtube.com/watch?v=vMZ7tK-RYYc

[19] https://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf

[20] https://www.nvidia.com/content/dam/en-zz/Solutions/Data-Center/documents/NVIDIA-Kepler-GK110-GK210-Architecture-Whitepaper.pdf

[21]https://www.nvidia.com/content/dam/en-zz/Solutions/design-visualization/technologies/turing-architecture/NVIDIA-Turing-Architecture-Whitepaper.pdf

[22] https://developer.nvidia.com/pascal

[23] https://devblogs.nvidia.com/inside-pascal/