ISSN:0975-9646

GPGPU in the Most Avid Communication Medium: Emails

Prachi Goyal Juneja, R.K.Pateriya

CSE, M.Tech Scholar
Maulana Azad National Institute of Technology
Bhopal(M.P) India-462001

***Abstract*- Email communication has gained lots of importance in the world lately. With the boundaries increasing in dimensions, it has become tough to rely on physical medium of communication due to delays. Internet being available anywhere and everywhere makes it convenient and gives portability and flexibility to work from multiple locations. GPGPU's have decreased the processing time and made the work faster and more usable. To gain most out of the GPGPU's, one needs to understand the architecture of the same and also the terminologies used and the programming languages used.**

***Keywords* - GPGPU, graphics processing unit, parallel processing, parallel programming, OpenCL, Cuda.**

I. INTRODUCTION

With the increase in computer usage around the globe and dependency structure routing towards computer at each stage, it is utmost important to make full and proper use of the CPU and provide maximum efficiency to the user. One can see the use and application of software devices in every walk of life: Science, Medicine, Education, Astronomy, Molecular Physics, and Neural Networks. With the increase in applications, the complexity increases and for coping up with a world where time is money, one needs to understand the importance of each and every second rather every nano-second. The information is nowadays stored in digital memories in enormous sizes that is rapidly growing. The challenge is to extract the relevant piece of information quickly. Hence we see the growth of data intensive applications [1][2]. Such applications can benefit from parallel computing in improving performance and for better data management.

Parallel Computing is use of multiple computer resources at the same time to solve a computation problem. In parallel computing a problem is broken down into various parts and each part is solved concurrently. A series of instructions from each part are executed simultaneously on different processors and an overall control mechanism is used.

In the real world where most of the events are uncertain it is most likely to use parallel computation instead of serial. One can see the wide use of parallel computing in the field of : galaxy formations, climate change, rush traffic hours, weather, defense, Geology, data mining, financial modeling, graphics and the list goes on. Some of the major benefits of using parallel computing are:

I. Helpful in solving larger problems
II. Provides concurrency
III. Use of non local resources
IV. Saves time and money

II. EVOLUTION OF PARALLEL COMPUTING

For a long time parallel computing [2] went unnoticed, as many thought it will never cope up with the computing scenario. During the 19th century parallel computing started evolving; [3] one can go back to the 1980's for the beginning of parallel computing. The major reasons for parallel computing evolution during this time are:

I. Advances in the hardware
II. Growth of research and development
III. Algorithms supporting parallel machines

Very rapidly we saw that over the years parallel computing slowly and steadily evolved and took shape. It was since 2010 that GPU became an important component of the PC, initially for good resolution in games, and media. Till now data used to flow from the CPU to the GPU. Then gradually there were developments in the data transport technologies and the one way data floe became two way data flow. Thus very rapidly the technology progressed in the recent past and the GPU usage was actively improved and increased from just a drawing tool to a very efficient processing unit.

III. WHAT IS GPGPU?

GPGPU [2] stands for General-purpose computing on graphics processing units. GPGPU often termed as GPU, is the use of graphic processing unit along with the CPU to fasten scientific, engineering and enterprise applications. GPU accelerated computing shows unmatchable increase in application performance. This is done by allocating the compute intensive portions of the task or application to the GPU, while the remaining code runs on the CPU.

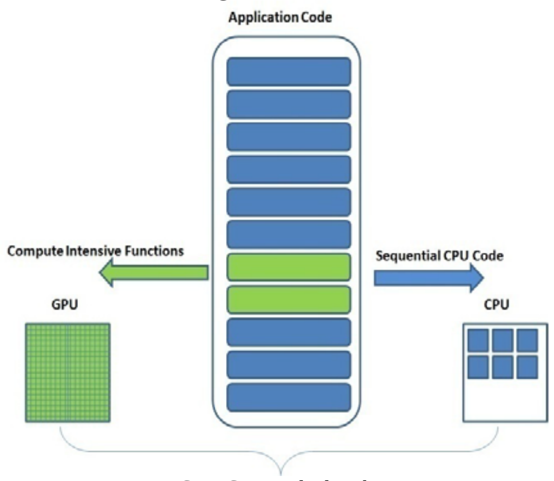

Fig. 1 CPU GPU code distribution

A CPU has few cores to carry out sequential serial processing whereas in case of GPU there are thousands of cores, smaller and more efficient. These are designed to handle multiple tasks simultaneously. This is why GPU's can run algorithms 100 times faster than CPU's.
Algorithms that can be executed on GPGPU's should have the following two properties:

i. Data Parallelism: Data parallel means that an operation can be performed on different data elements simultaneously by a processor.
ii. Throughput Intensive: Since there are high computations involved, the amount of data elements is in abundance and the algorithm will process them in parallel.

The best suited and widely used area of GPGPU's remained to be pixel- based applications such as video and image processing, but with advancement in the GPGPU field applications that have heavy numerical computing and linear algebra operations are also well suited for GPGPU technology.
Some areas where GPGPU based applications are widely used are: Research, Supercomputing, Defense and Intelligence, Finance, Fluid Dynamics, CAD, Automations, Animation, Simulation, Media, Oil and Gas, Biology, Chemistry and the list is endless.

IV. GPGPU ARCHITECTURE

A GPGPU consists of large number of streaming multiprocessors(SM), which can be increased in number. These SM's are connected to the Interconnection network through which they can access multiple memory controllers. [4]

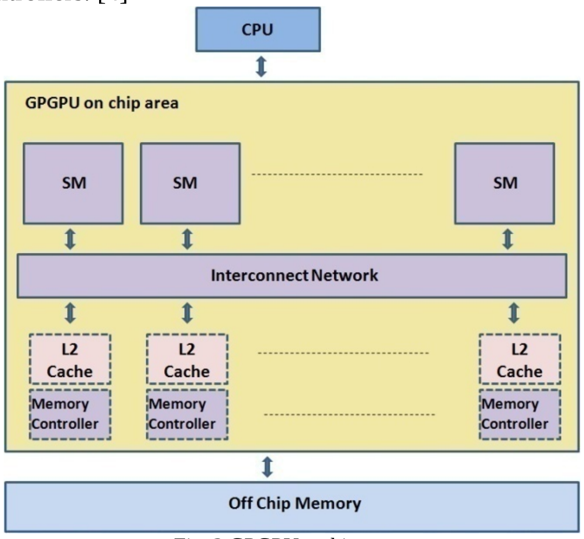

Fig. 2 GPGPU architecture

When an application code is given to the processor, the serial code is executed by the CPU and the compute intensive code is passed on to the GPU where stream processing occurs in parallel.
Each SM contains: [5]

- Thousands of registers to be allocated to execution threads
- Caches
  - shared memory
  - constant cache
  - texture cache
  - L1 cache
- Warp schedulers
- Stream Processors to carry out operations
-

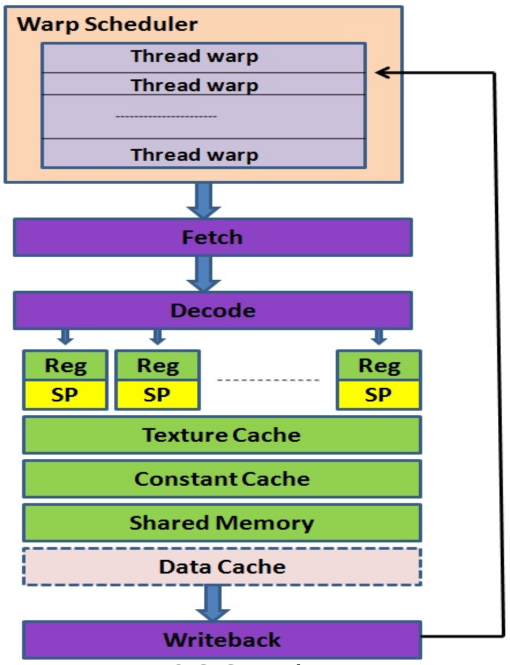

Fig. 3 GPGPU architecture

Figure 3 depicts a general architecture of GPU, some high end GPU's can have additional parts as well.

V. GPGPU: BASICS AND ITS FEATURES

GPGPU's are great for data parallelism. They are designed for tasks that are highly parallel. The GPU architecture as discussed above is ALU heavy, i.e., there is lots of compute power and multiple vertex and pixel pipelines are available. GPU's are designed to stream data, they hide memory latency. The GPGPU terminology is discussed here:

- Arithmetic Intensity: It refers to computations or math operations.
- Streams: A set of records that have to undergo the same type of computation. They provide data parallelism.
- Kernels: Functions applied to each element in the stream. The elements in a stream should be independent of each other.

The applications targeted to be run on GPU should have:

- High parallelism
- High arithmetic intensity
- Minimum dependency between data elements
- Huge data sets
- Lot of work to be done without CPU intervention

Before writing code for any application an algorithm is set into place. These CPU algorithms are to be mapped with the GPU. The various mapping constructs are:

- Basics: [6]
  - Arrays/ streams → Textures
  - Parallel loops→ Quads

- o Loop body → vertex + fragment program
- o Output arrays → render targets
- o Memory read → texture fetch
- o Memory write →frame buffer write

❖ Controlling the parallel loop:[6]

- o Rasterization = Kernel Invocation
- o Texture Coordinates = Computational Domain
- o Vertex Coordinates = Computational Range

As discussed the GPU's follow the SIMD (Single Instruction Multiple Data) [7] execution model. If you take a closer view at the memory model it follows shared memory access. At the programming model level we see a wide use of C/C++ along with the extensions to do programming of GPGPU's. CUDA and OPENCL are two famous programming forms that support the same.

Each technology comes with its ups and downs and so is the case with GPGPU programming. It has some advantages as well as disadvantages, though the advantages take a head over the disadvantages, but it's necessary to be aware of all the pros and cons .[7]

Advantages:

- ➢ Fast
- ➢ Energy efficient
- ➢ Cheap
- ➢ Load sharing of CPU

Disadvantages

- ➢ Comparatively hard to program
- ➢ Not all algorithms might necessarily have speedup
- ➢ The industry needs to mature more

VI . Parallel Programming Languages

GPGPU's were designed mainly for graphics and later the multi-core concept spread widely to computational problems. GPU's can process independent pieces of tasks in parallel. GPGPU's are mainly used for stream processing, i.e., similar computations to be applied on different set of data in parallel. One important point to be noted is that GPGPU's never have stared data,[8] because processing is performed in parallel and independently. Since GPGPU is used widely for applications that are critical in nature, hence they are bound to have high arithmetic intensity. If it's not so then the delay in accessing memory will result in computational delay. The computations performed by the GPU are generally handled by the CPU [9]. Parallel computing systems today have more than one CPU or GPU in a single machine for faster computation and performance. In the beginning of the CPU + GPU systems, languages with an extension of C like Brook and Cg were used [10].

The most widely used GPGPU computing language present today is OpenCL (Open Computing Language). OpenCL provides framework that is usable for writing programs over heterogeneous platforms having CPU's, GPU's and other processors. It includes a programming language to write kernels for computation. OpenCL has a feature available to support task parallel programming patterns, which are required for multicore architecture.[9]

Another dominant framework available for parallel computing is Nvidia's CUDA(Compute Unified Device Architecture). The CUDA platform is available for users through libraries and extension to standard languages like C, C++ and Fortran. The CUDA programs contain the conventional language host code and GPU device functions [11]. In CUDA the programmer is not responsible for thread management, it is done implicitly.

VII. Comparative Study: OpenCL And Cuda

With the advent of parallel programming languages, one needs or make a decision about choosing the language as per the requirement and usability. Cuda and OpenCL are similar in many respects like both are focused on data parallel computation model [12]. Both use C based languages for device programming. Apart from the similarities there is a huge range of differences also.

	OPENCL	**CUDA**
What is it?	Hardware architecture, programming language, API, SDK and tools	Open API and language specification
Open technology?	Not open	Open and royalty free
Started in?	2006	2008
SDK Vendor	Nvidia	Implementation Vendor
Multiple vendors	No	Yes
Vendors	Nvidia	Nvidia, Apple, IBM, AMD
Device Support	Only Nvidia	Heterogeneous device support
Terminology(Execution) [12]	NDRange	Grid
	Workgroup	Threadblock
	Workitem	Thread
	Global ID	Thread Id
	Block Id	Block Index
	Local ID	Thread Index
Terminology (Memory)	Host memory	Host memory
	Global Memory	Global or device memory
	Global memory	Local memory
	Global memory	Texture memory
	Local Memory	Shared memory
	Private memory	Registers
Programming		
Language versions	Base language versions are defined [15]	Only C and C++

	OPENCL	CUDA
		features supported
Work items access	Through built in functions	Through built in variables
Vector types	Vector types, built in operators and functions, literals	Vector types defined, no operators or functions
Memory[13] [14]	Asynchronous memory copying and pre fetch functions	No asynchronous memory copying and pre fetch functions
C++ features	Not supported	Limited features Supported
API [14]	OPENCL API	CUDA Driver API
Setup [15]	-Initialize platform -Get devices -Choose device -Create context -Create command queue	-Initialize driver -Get device(s) -(Choose device) -Create context
Device and host memory buffer setup[13]	-Allocate host memory -Allocate device memory for input -Copy host memory to device memory -Allocate device memory for result	-Allocate host memory -Allocate device memory for input -Copy host memory to device memory -Allocate device memory for result
Initialize kernel	-Load kernel source -Create program object -Build program -Create kernel object bound to kernel function	-Load kernel module -(Build program) -Get module function
Execute the kernel	-Setup kernel arguments -Setup execu6on configuration -Invoke the kernel [15]	-Setup kernel arguments -Setup execu6on configuration -Invoke the kernel
Copy results to host	Copy results from device memory	Copy results from device memory
Cleanup	Cleanup all set up above	Cleanup all set up above

VIII . CONCLUSION

Advancements in the field of GPGPU's have led to the wide use of GPGPU's in high intensity arithmetic computations. It has led to the load sharing of the CPU and has made it more easy and organized to allocate high computation oriented task to the GPU and rest to the CPU. Most of the algorithms that are serial in nature can be converted to parallel form and run on the GPU. Everybody today, especially in the computation heavy application industry is working towards using parallel architecture to perform operations and complete tasks faster. Use of GPGPU's has made our systems more efficient and the use of processor is made to the fullest.

REFERENCES

[1] Mario Cannataro , Domenico Talia , Pradip K. Srimani "Parallel data intensive computing in scientific and commercial applications", Parallel Computing, 2011, ELSVIER, p 02.

[2] J. Nickolls and W. J. Dally, "The GPU Computing Era," *IEEE 2010 Micro*, vol. 30, pp. 56–69.

[3] David E. Womble, Sudip S. Dosanjh, Bruce Hendrickson,Michael A. Heroux, Steve J. Plimpton, James L. Tomkins,David S. Greenberg, "Massively parallel computing: A Sandia perspective", Parallel Computing 1999, E:SEVIER,pp 01-03.

[4] Jingweijia Tan , Yang Yi , Fangyang Shen , Xin Fu "Modeling and characterizing GPGPU reliability in the presence of soft errors" Parallel Computing 2013, ELSEVIER IEEE 2007, pp 01-02, 521-523.

[5] Nicholas Wilt , "8 Streaming Multiprocessors" Pearson Education Inc, 2012.

[6] "General Purpose Computation on Graphics Processors (GPGPU)" Website: "http://graphics.stanford.edu/~mhouston/public_talks/R520-mhouston.pdf", [Accessed: April 14, 2014]

[7] Xiaoqing Tang, "Introduction to General Purpose GPU Computing", University of Rochester 2011, p 03.

[8] General-purpose computing on graphics processing units Website: "http://en.wikipedia.org/wiki/General-Purpose_Computing_on_Graphics_Processing_Units", [Accessed: May 05, 2014].

[9] Mohammad Reza Selim , Mohammed Ziaur Rahman , "Carrying on the legacy of imperative languages in the future parallel computing era", ELSEVIER, 2013, pp 02-07.

[10] J. Diaz, C. Muñoz-Caro and A. Niño" A Survey of Parallel Programming Models and Tools in the Multi and Many-core Era" IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE 2011, pp 01-05.

[11] "GPU COMPUTING: THE REVOLUTION ", Website: http://www.nvidia.com/object/cuda_home_new.html [Accessed: April 4, 2014].

[12] Jianbin Fang, Ana Lucia Varbanescu and Henk Sips "A Comprehensive Performance Comparison of CUDA and OpenCL" pp 01-02.

[13] John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone, and James C. Phillips "GPU Computing" IEEE, 2008, pp 8-17.

[14] R. Amorim, G. Haase, M. Liebmann, and R. Weber dos Santos, "Comparing CUDA and OpenGL implementations for a Jacobi iteration," Presented in IEEE Conference 2009, pp. 22–32.

[15] The Khronos OpenCL Working Group, "OpenCL - The open standard for parallel programming of heterogeneous systems." Weblink: http://www.khronos.org/opencl [Accessed on April 5, 2014].