A Comparative Study of GPU Computing by using CUDA and OpenCL

Asad Mohhamad Vikram Garg

ABSTRACT
Parallel computing becomes a need to perform task as soon as possible. This can be done in two way improve hardware or use parallel programming language i.e. improve software. Improvement in the hardware is costlier solution compared to software solution. So we have two basic heterogeneous parallel languages CUDA and OpenCL which run on both CPU and GPU according to necessity. When program does not contain high parallelism it works on CPU which contains less number of cores. On other hand program contain high degree of parallelism so each independent code runs on separate core of GPU. This paper gives the basic idea of the parallel computing and how these carried out. Explain the working of both parallel language CUDA and OpenCL with their detailed architecture. In the last section comparison of both languages is described.

Keywords
Parallel Computing, GPU, GPGPU, CUDA, OpenCL.

1. INTRODUCTION
Today's world everything would be quick and efficient. Performing task in a serial manner on silicon based processor chip are constraint by the speed of light and thermodynamics law so processing speed can be increased up to certain level. So other solution of this is "Go Parallel" means perform the task in the parallel manner this can boost up the speed. Parallelism can be achieved in two way hardware solution and software solution. Hardware solution is one where thousands of CPU's are used working with co-ordination to each other to solve the big problem. Some of the hardware are NUDT tianhe, IBM sequaio and PARAM supercomputer. Hardware solution is not commonly used now because they are specific to particular hardware and costly. So software solution is the cost effective solution which exploit the existing hardware. Software solution can use CPU as well as GPU. In the system we have limited number of CPU's are presents due to this our parallelism is limited. On other hand GPU scope is very wide because it consists of number of cores or ALU which helps to obtain parallelism. Here the task is divides into independent subtask and process them on to different cores.

2. GENERAL PURPOSE GRAPHICS PROCESSING UNIT
GPU stands for graphics processing unit traditionally it was used for high definition video and gaming purpose. Now a day GPU consists of thousands of ALUs and core. These core or ALUs are used to make GPU as programmable. These programmable capacity of a GPU make them general purpose and known as GPGPU. Figure 1 shows the AMD HD 6450 GPU. GPU consist of global memory which is common to all and local memory dedicated to each compute unit. In the figure we shows the two compute units and each compute unit consist of 16 stream unit. Each stream unit consists of four ALU and one special function unit. So program is divided into threads and provides the separate ALUs to operation.

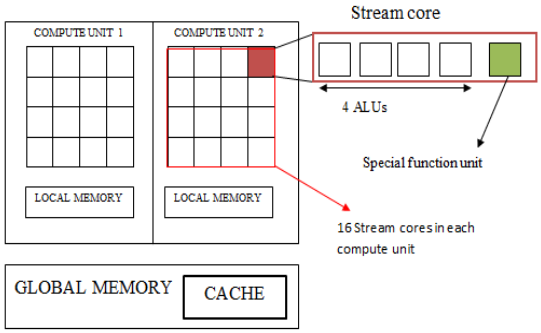

Figure 1: GPU Architecture of AMD HD 6450

3. OPEN COMPUTING LANGUAGE
OpenCl is a parallel programming language standardized by khronous group. OpenCL is a portable heterogeneous computing language means it can be operated on CPU's, GPU's, DSP's and hardware. As a name stands it is open for all vender means by doing small changes it can be run in any type of hardware. Figure 2 shows that OpenCL is a heterogeneous computing language means serial function is execute on the host CPU while function with parallelism arrives it is execute on parallel kernel of GPU.

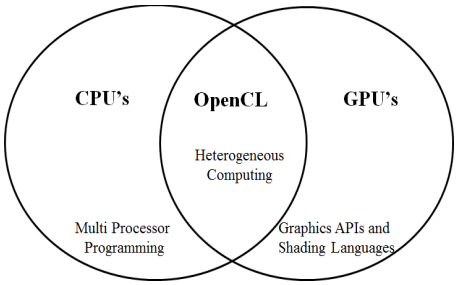

Figure 2: Inspiration for OpenCL

Architecture of OpenCL Shown in Figure 3 here the smallest individual unit is called work item. On other hand group of work item is called work group. If program consist of parallelism than host called the kernel for execution of a code in parallel manner. According to degree of parallelism work items are allocated each parallel code which execute parallel. The work of NDrange is to instruct the kernel about the work group and other information. In OpenCL we have various range of memory like as private memory, local memory, global memory and constant memory. Private memory and local memory both are stored in the GPU chip means local to work group. Only work item of the work group can access these two memories. On other hand constant memory and global memory are GPU memory can be used by any of the work group. Accessing speed of these memory can be arrange in ascending order is as follow (means less access time first) Private memory, Local memory, Constant memory and Global memory.

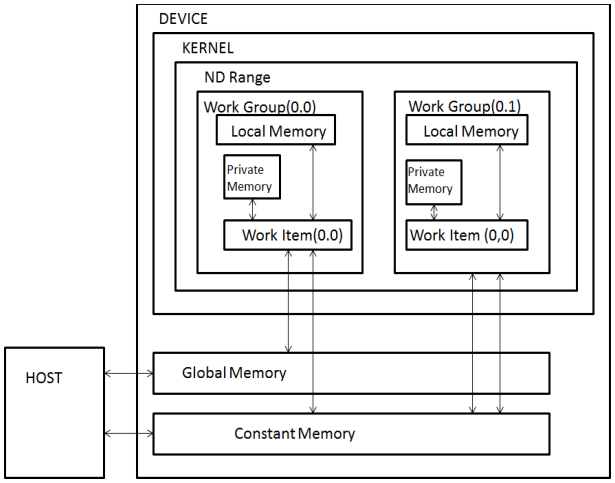

Figure 3: Architecture of OpenCL

The one of the significant advantage of OpenCL is it is a cross platform language means by small changes it can run on any platform. Another is it is heterogeneous language means serial code run on CPU and parallel code runs on GPU.

4. COMPUTE UNIFIED DEVICE ARCHITECTURE

CUDA is a parallel computing platform implemented by NVIDIA. CUDA stands for Compute Unified Device Architecture. CUDA is vender specific means it is operated on only NVIDIA graphics only. CUDA is a specific to only NVIDIA so it knows the architecture of the NVIDIA very well due to this it gives us very efficient results. Figure 4 shows the basic architecture of the CUDA. This is similar to the OpenCl rather than basic terminology which is described in the comparison part.

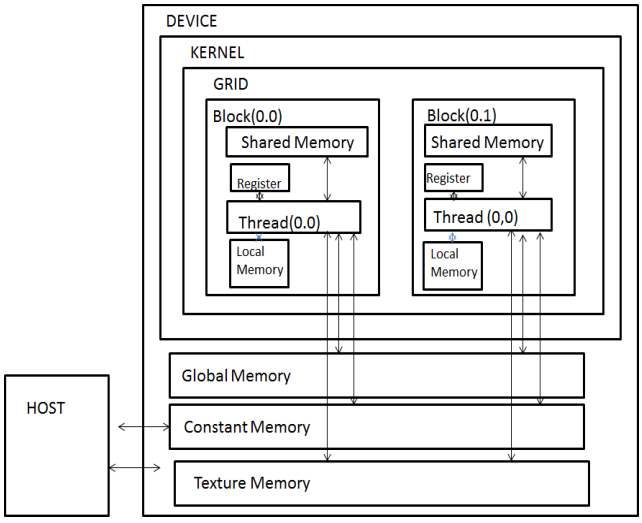

Figure 4: Architecture of CUDA

CUDA gives the faster result because it is dedicated to the NVIDIA platform so code is specific to that platform. It is also a heterogeneous language which runs on both CPU as well as GPUs.

5. COMPARISON BETWEEN CUDA AND OPENCL

CUDA and OpenCL both are used for parallel platform. The architecture of the CUDA language is similar to the OpenCL language it differs in the terminologies. The terminology comparison is described in the Table 1.

Table 1: Comparison of basic termonoly of OpenCL and CUDA

Terminology	OpenCL	CUDA
Smallest Unit	Work Item	Thread
Group of Smallest Unit	Work Group	Block
Connectivity to Kernel	NDRange	Grid
Memory which can only use by group of smallest unit	Local Memory Private Memory	Register Memory Shared Memory Local Memory
Memory can be used to any group and any where	Constant Memory Global Memory	Constant Memory Texture Memory Global Memory

Other than architecture these two languages are differ in most of the cases. The detailed comparison of OpenCl and CUDA are described in the Table 2.

Table 2: Comparison between CUDA and OpenCL

Parameter	CUDA	OpenCL
Vender	Vender Specific	For all Vender
Speed	It is vender specific so dedicated platform is provided to CUDA so it runs faster	It is open to all so runs slower comparatively
Codes	Specific code for NVIDIA	It is similar to C code

6. CONCLUSION

Parallel processing is the most efficient way to solve the problem. This divides the whole problem into independent sub problems and executes them parallel. There are two basic parallel programming languages which can execute the program parallel CUDA and OpenCL. CUDA language is platform dependent so it will support by only NVIDIA platform and coding is little bit complex. On other hand OpenCL is open to all supported by most of the GPUs and coding is similar to C language which is quite easier to understand. So OpenCl is a good choice to perform parallel operation.

7. REFERENCES

[1] J. D. Owens, D. Luebke, N. Govindaraju, M. Harris, J. Kru¨ger, A. E. Lefohn and T. Purcell B, " A survey of general-purpose computation on graphics hardware", Comput. Graph. Forum, vol. 26, no. 1, pp. 80–113, 2007.

[2] K. Fatahalian and M. Houston, "A closer look at GPUs," Communications of the ACM, Vol. 51, No. 10, October 2008.

[3] Z. Fan, F. Qiu, A. Kaufman, S. Yoakum-Stove, "GPU Cluster for High Performance Computing,", in Proc. ACM/IEEE conference on Supercomputing, 2004.

[4] D. Tarditi, S. Puri, and J. Oglesby, "Accelerator: Using data-parallelism to program GPUs for general-purpose uses", in Proc. 12th Int. Conf. Architect. Support Program. Lang. Oper. Syst., pp. 325-335, Oct. 2006.

[5] John D. Owens, Mike Houston, David Luebke and Simon Green," GPU Computing Graphics Processing Units-powerful, programmable, and highly parallel-are increasingly targeting general-purpose computing applications", In the procd. Of IEEE Xplore, Vol. 96, no. 5, May 2008.

[6] John Nickolls, William J. Dally NVIDIA, "THE GPU COMPUTING ERA", In the procd. Of IEEE Computer Society, page no. 56-69, March 2010.

[7] Danilo De Donno, Alessandra Esposito, Luciano Tarricone, and Luca Catarinucci, "Introduction to GPU Computing and CUDA Programming: A Case Study on FOlD" In the procd. Of IEEE Antennas and Propagation Magazine, Vol. 52, No.3, June 2010.

[8] NVIDIA Corporation Technical Staff, NVIDIA CUDA Programming Guide 2.2, NVIDIA Corporation, 2009.

[9] M. Harris, S. Sengupta, and J. D. Owens, "Parallel prefix sum (scan) with CUDA", in GPU Gems 3, H. Nguyen, Ed. Reading, MA: Addison-Wesley, pp. 851–876, Aug. 2007.

[10] J. Nickolls et al., ''Scalable Parallel Programming with CUDA,'' ACM Queue, vol. 6, no. 2, pp. 40-53, 2008.

[11] Ching-Lung Su, Po-Yu Chen, Chun-Chieh Lan, Long-Sheng Huang, and Kuo-Hsuan Wu, "Overview and Comparison of OpenCL and CUDA Technology for GPGPU" In the procd. Of IEEE, Page no. 448-451, 2012.

[12] Misic, M.J., Durdevic, D.M. ; Tomasevic, M.V., "Evolution and trends in GPU computing" In the procd. Of IEEE 35th International Convention MIPRO, Page no. 289-294, 21-25 May 2012.

[13] Jääskeläinen, Pekka O., de La Lama, Carlos S, Huerta, Pablo, Takala and Jarmo H, "OpenCL-based design methodology for application-specific processors", In the procd. Of IEEE International Conference on Embedded Computer Systems (SAMOS), February 17, 2011.

[14] Benedict R Gaster, LEE Howes, David Kaeli, Perhaad Mistry and Dana Schaa, "Hetrogeneous Computing with OpenCL" Book publised by Elsevier, 2012.

[15] Aaftab Munshi, Benedict R. Gaster, Timothy G. Mattson, James Fung and Dan Ginsburg, "OpenCL Programming Guide", Book publised by Pearson Education, 2012.