Volume 5, Issue 11, November 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Solving a Complex Problem with the Help of SIMD using GPU

Himanshu Nayak, Rajesh Tiwari
CSE, SSTC, CSVTU, Bhilai,
Chhattisgarh, India

***Abstract*— In this paper we intend to analyse and compare the computation time of single processor machines and multi core machines. CUDA library and GPU with multi-cores is used to achieve parallelism. The operation is to compute multiplication of two matrices. The process is repeated for various matrix sizes on both versions i.e. serial and parallel. The results are analysed for various matrix sizes**.

Keywords*— Compute Unified Device Architecture, Graphical Processing Units, Parallel Processing Languages.

I. INTRODUCTION

Parallel processing is widely used in the field of computer science in recent decades, it is though not new in day to day life. In real life when there is some problem or task which is not possible for a single person, we tend to co-operate and finish the task. Similarly, in computer science when any large commutation problem and if it has multi-core machine, and divide it into small modules, then allocate these modules to separate processors. Those modules executed on different processors and then the overall result is combined to generate the final output. The processing power of a single processor has constantly been increasing researchers has worked very hard to make a single processor with very high capacity of computing power. The limit of a single processor has been defined and about to reach to its peak. This is because there are certain constraints for a processor's computation power viz design by its physical implementation size, heating issues, and inherent bottlenecks. But this does not limit the speed requirements, day by day the requirement of high speed processors is increasing. So to solve such problems there this phenomenal concept of Parallel Processing is being used.

CUDA is a library provided by NVIDIA, it provide extended functionalities in C language by adding CUDA specific functions.

Here studied the approaches and later developed a methodology to compare the execution time for matrix multiplication problem.

Here execute a CUDA program to compare process times on a GPU and a CPU for a range of task size. The hardware contents NVIDIA GEFORCE GT635M GPU and the duel-core Intel i5 CPU processors. The operation is to compute multiplication of two matrix. The process is repeated for various matrix sizes on both the environments serial and parallel. The result is then analysed and as a conclusion found that from smaller size matrix up to a threshold value the CPU performs faster than GPU. After the threshold when the matrix size is increased, The performance of GPU is always better than CPU's. This is because when execute the process in GPU need to transfer the data and instruction to GPU from CPU, and after computation the result is again copied to CPU memory from GPU. These two data transfer takes time, when the matrix size is small this communication time becomes significant as the execution time is very small. In case of large matrix size the data transmission time is negligible as compared to execution time so GPU performs better than CPU.

II. CUDA AND GPU

CUDA (Compute Unified Device Architecture) is a library provided by NVIDIA to execute processes in parallel. This makes user able to communicate with the hardware directly. There are CUDA specific functions or methods defined which meant to run on CUDA library only. These are used along with C and C++ programming language. To convert a single processor specific program into CUDA capable programs the programmer needs to modify it accordingly. The CUDA program is generally divided into two parts: the main program executes in the CPU, whereas the parallel portion of the program is executed in GPU. This GPU part is called by main program and data is sent to GPU for execution where the instructions are executed on the data, after the calculation result is sent back to CPU.

GPU (Graphics Processing Unit) was primarily developed to fulfil the need of algorithms used in computer graphics. It has hundreds of cores which were able to execute multiple threads simultaneously. Later it was proposed that this technology can be useful for non graphic process also, if one can divide a single process into multiple threads and distribute them to multiple processors, the overall computation time can be reduced drastically. There are several types of memory present in the GPU like device memory, shared memory, constant cache, texture cache, and registers. To manipulate data among these memory and to use the multiple cores to their programmers must write the CUDA programs very carefully.

CUDA Memory Types

There are various types of memory present in the CUDA architecture, we will discuss all these memory organization in brief.

Shared Memory: Each Block has a Shared memory which is shared by all its threads for communication within the block. All threads can read and/or write in this memory region. It has size smaller than that of Global Memory, generally it is around 50 to 100 MB. The hardware which we have used for this implementation has 49125 Bytes of Shared Memory. The total number of threads that will be able to execute simultaneously is determined by the size of shared memory.

Registers: This is fastest accessible memory present in the GPU. Each of the threads are given one set of register memory, which is used by threads to store very frequent used data like counters etc. The cores directly accesses the data from registers only, first the data is brought to registers then only the operations are performed in similar approach as the single processor machines. In the GPU hardware we are using we have 32768 number of registers available per block.

Basic Units of CUDA

The whole architecture of CUDA can be viewed as consist of three major components parts namely Grid, Block and Threads in hierarchal fashion (Figure 1). We will discuss about each one of these in this subsection.

Grid: A Grid is collection of several Blocks. All the threads in the Block runs same Kernel. A grid is started in synchronous form in the CPU, but there can be multiple Grids running at the same time. These Grids cannot be shared in multi-GPU systems.

Block: A Block is a logical unit which contains multiple coordinating threads. Blocks are no sharable among multiprocessor same as the Grids. All the Blocks belonging from same Grid, executes the same code. blockID is a built-in variable which is used to identify each Block uniquely within a Grid. In general there are 65,535 Blocks present per Grid.

Thread: Threads are the basic unit of CUDA architecture. The thread runs on separate cores of multiprocessors, but they are not restricted to one core as Grids and Blocks. Threads are identified by threadIdx, which can be 1D, 2D or 3D. Each thread can have a pair of Register memory for fast access.

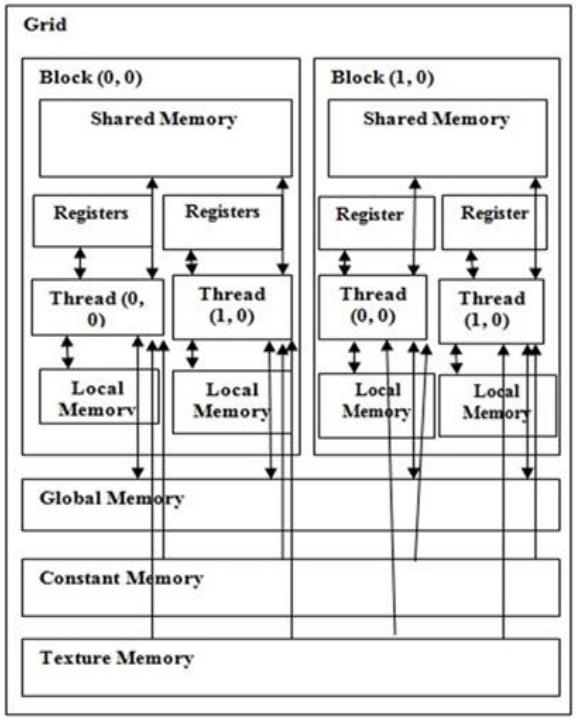

Figure 1: CUDA Units and Memory Type

III. LITERATURE SURVEY

S. No.	Title	Author/Publication year	Algorithm Analyse	Conclusion
1	GPU-to-GPU and Host-to-Host Multipattern String Matching on a GPU	XinyanZha and SartajSahhni. June 2013, IEEE	1. AhoCorasick 2. BoyearMoore	In GPU-to-GPU and Host-to-HostAho-Corasick Algorithm Preform better.
2	A Coprime Blur Scheme for Data Security in Video Surveillance.	Charistopher Thorpe, Feng Li, Zijia Li, Zhan Yu, David Saunnders, and JingyiYu.	1.GCD(Grates Common Deviser) 2. Bezout matrix	Implemented GPU based processing method that produced deblurred image.

		December 2013, IEEE		
3	K-Means for Parallel Architectures Using All-Prefix-Sum sorting and updating Steps	Kai J. Kohlhoff, Vijay S. Pande, and Russ B. Altman. August 2013, IEEE	1. MATLAB R2008b Software package.	Comparison Speedup between GPU and MATLAB R2008b S/W package. GPU perform better than the MATLAB S/W package.
4	Designing Efficient Sorting Algorithms for Manycore GPUs	NadathurSatish, Mark Harris, Michael Garland. January 2010, IEEE	1.Redix Sort Algorithm 2.Merg Sort Algorithm	Both Algorithm Performed operation on GPU. Radix Sort is faster than Merg Sort on GPU.

IV. PROBLEM IDENTIFICATION

From the study in literature survey we can conclude that the multi processor machines are performing better as compared to the single processor machine in terms of execution time. In this paper we are experimenting with matrix multiplication problem and will try to come to the conclusion that if the multi processor machine performs better than the single processor.

V. IMPLEMENTATION

Simulation Environment: The system environment used in implementation are following:

System specification:

- Operating System: Windows 7 Ultimate
- Visual Studio 2010
- CUDA Tool Kit 7.0

GPU Specification:

GPU which we have use for our implementation has following specification:

- Device Name: NVIDIA GEFORCE GT635M
- Total amount of global memory: 2048 MB
- Number of Multiprocessors: 2 (48 CUDA Core/MP)
- Number of Streaming prospectors cores: 98 CUDA Core
- GPU Clock rate: .95 GHz
- Memory Clock rate: 900 MHz
- Total amount of constant memory: 65536 Bytes
- Total amount of shared memory per block: 49125 Bytes
- Total number of register available per block: 32768
- Warp size: 32
- Maximum number of thread per multiprocessor:1536
- Maximum number of thread per block: 1024
- Max dimension size of a thread per block: (1024,1024,64) (x, y, z)
- Max dimension size of a thread per grid: (65535, 65535,65535) (x, y, z)
- Texture alignment: 512 bytes

We ran the simulation with predefined simulation parameters and got the results tabulated in above table. Those simulations were executed for 30 times for each simulation configuration, mean value of those results are taken as the final value of that particular simulation.

Table 1: CPU and GPU Avg. Time

S. No.	Size of Matrix	Average CPU Computation Time (in ms)	Average GPU Computation Time (in ms)
1	5x5	.00333	.03310
2	10x10	.01531	.01713
3	15x15	.03114	.02308
4	20x20	.05127	.02950
5	25x25	.08555	.03907
6	30x30	.12138	.05386

V. RESULTS AND ANALYSIS

In the graph shown in Figure 2 we can observe that when the matrix size is smaller, then time taken by GPU for multiplying the matrixes is more than that of CPU. After a threshold value of matrix size, time taken by CPU is greater than the time taken by GPU. The reason for this is, when the matrix is small then the time for data set transfer from CPU

to GPU and again transferring the result from GPU to CPU is considerable as compared to the total execution time. So when small matrixes are given as input for multiplication the multi processor machine spends more time in data transfer than computation, meanwhile the CPU can compute the result in less amount of time. In the experiment when we increase the size of matrixes gradually the time taken for CPU goes higher than GPU. This is because now the time taken for data set and result transfer between CPU and GPU is far smaller than the computation time. The CPU now takes more time than the multi processor machine.

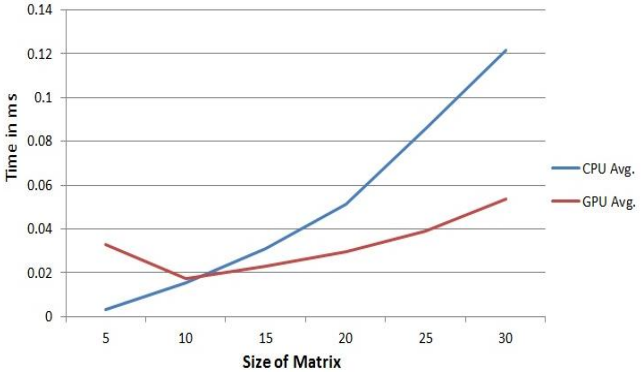

Figure 2: Comparison Graph

VI. CONCLUSION

We performed the simulation for matrix multiplication problem on both the environment and obtained the results which are discussed and analysed in the previous section. The resultant graph showed that using multi-core system does not gives better performance than single processor system always. It depends on the input type and size of the input as well as that which type of environment is suitable for solving the specific problem.

REFERENCES

[1] Z. Xiyan, S. Sartaj "GPU-to-GPU and Host-to-Host Multipattern String matching on a GPU" IEEE Transaction on Computers, Vol. 62, No. 6, JUNE 2013.

[2] T. Christopher, L. Feng, L. Zijia, Y. Zhan, S. David,Y. Jingyi " A Coprime Blur Scheme for Data Security Video Surveillance" IEEE Transaction On Pattern Analysis and Machine Intelligence, Vol. 35, No. 12, DECEMBER 2013.

[3] K. Kai J., P. Vijay S., A. Russ B.,"K-Means for parallel Architecture Using All-Prefix-Sum Sorting and Updating Steps", IEEE Transaction on Parallel and Distributed System. Vol. 24, No. 8, AUGUST 2013.

[4] G. Jayshree, P. Jitendra, K. Madhura, B. Amit "GPGPU Processing in CUDA Architecture" Advanced Computing: An International Journal (ACIJ), Vol.3, No.1, January 2012.

[5] S. Nadathur, H. Mark, G. Michael "Designing Efficient Sorting Algorithms for Manycore GPUs" IEEE Explorer, January 2010.

[6] K. J. edrzej, P. T. Eric, C. P. Lance "Real-Time Stereo Matching on CUDA Using an Iterative Refinement Method for Adaptive Support-Weight Correspondences" IEEE Transactions on Circuits and Systems for Video Technology, Vol. 23, No. 1 January 2013.

[7] W. Wei, Q. Fengbin, H. Wangquan, W. Shanshan, "CUDA's Mapped Memory to Support I/O Functions on GPU" Tsinghua Science and Technology ISSN 1007-0214 05/10, PP588-589, Vol. 18, No. 6, December 2013.

[8] N. Quang M., VinhDang, K. Ozlem, E. Esam,"Parallelizing Fast Multipole Method for Large-Scale Electromagnetic Probleams Using GPU Clusters" IEEE Antennas and Wireless Propagation Letters, Vol. 12, 2013.

[9] A. BriffaJohann,"A GPU Implementation of a MAP Decoder for Synchronization Error Correcting Code" IEEE Communications Letters Vol.17, No. 5, May 2013.

[10] B. William, U. Gray, M. Graham, "Applicability of GPGPU Computing to Real-Time AI Solutions in Games" IEEE Transactions on Computational Intelligence and AI in Games, Vol. 5, No. 3, September 2013.

[11] David Kirk/Nvidia and Wen-meiHwu, 2006-2008 – "CUDA Threads"

[12] R. Farivar, D. rebilledo, E. Chan, and R. Campbell, "A Parallel implementation of K-means Clustering on GPUs, "pro. Int'l Conf. parallel and Distributed processing Techniques and Applications, 2008.

[13] M. Zechner and M. Granitizer, "Accelareting K-Means on the Graphics Processor via CUDA," Proc. First Int'l Conf. Intensive Applications and Services (INTENSIVE '09), pp. 7-15, 2009, doi: 10.1109/INTENSIVE.2009.19

[14] A. Aho and M. Corasick, "Efficient String matching: An Aid to Bibliographic Search, "Comm. ACM, vol 18,no. 6, pp. 33-340, 1975.

[15] R. Baeza-yates, "Improved String Searching," Software Practice and Experience, vol. 19, pp.257-271,189

[16] R. Boyer and J. Moore, " A Fast String Searching Algorithm, "Comm. ACM, vol. 20, no. 10, pp. 262-272, 1997.

[17] P Agrawal and P. Narayana, "Person De-Identification in Videos," IEEE Trans. Circuits and Systems for Video Technology, vol. 21, no 3, pp. 200-310, Mar. 2011.

[18] M. Naor and A. Shamir, "Visual Cryptography," Pro. Advances in Cryptology, 1995.

[19] R. Tiwari et al. , " The Efficient load balancing in the parallel Computer" , International Journal of Advanced Research in Computer and Communication Engineering , Vol. 2, Issue 4, April 2013 , pp 1667 – 1671 , ISSN : 2319-5940 .

[20] R. Tiwari et al. , "Analysis of Various Decentralized Load Balancing Techniques" International Journals of Recent and Innovative Trends in Computing and Communication (IJRITCC), Vol2 , Issue 11 , November 2014, pp 3360 – 3365 , ISSN: 2321 – 8169.

[21] Miguel C'ardenas-Montes," Accelerating Particle Swaram Algorithm with GPGPU," 19th International EuromicroConfrence on parallel, Distributed and Network-Base Processing, 2011

[22] Danlio De Donno et al., "Introduction to GPU Computing and CUDA Programing: A Case Study on FDTD," IEEE Antennas and Propagation Magazine, June 2010.

[23] S.A.A. Shalom, M. Dash, and M. Tue, "Efficient K-Means Clustering Using Accelerated Graphics Processors," Proc. 10th Int'l Conf. Data Warehousing and knowledge Discovery I. Song, J. Eder, and T. Nguyen, eds., pp. 166-175, 2008, doi: 10.1007/978-3-540-85836-2_16.

[24] H. Bai, L. He, D. Ouyang, Z. Li, and H. Li, "K-Means on Commodity GPUs with CUDA," Proc. WRI World Congress Computer Sceince and Information Eng., Vol. 3, pp.651-655,2009,doi:10.11.1109/CSIE.2009.491

[25] Nayak H., Tiwari R., Sharma M., Mehta K., "A Study of GPU and CUDA for SIMD Concept", International Journal of Emerging Research in Management & Technology, ISSN:2278-9359, Vol. 4, Issue 6, June 2015.

