ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

GPU Acceleration Using CUDA Framework

Pratul P Nambiar[1], V.Saveetha[2], S.Sophia[3], V.Anusha Sowbarnika[4]

[1] UG Student, Department of IT, Info Institute of Engineering, India.

[2, 4] Assistant Professor, Department of IT, Info Institute of Engineering, India.

[3] Professor, Department of ECE, Sri Krishna College of Engineering & Technology, India.

ABSTRACT: This paper deals with functioning and application of graphics processing units to general purpose computing and the high performance capability of a Graphics Processing Unit(GPU) using CUDA(Compute Unified Device Architecture) to do parallel computing. GPGPU which stands for General-purpose computing on Graphics Processing Units is the technique in which the GPU is employed to handle and perform computations that were previously handled only by the CPU. Parallel computing is a form of computation in which many calculations are carried out simultaneously, operating on the principle that large problems can often be divided into smaller ones, which are then solved concurrently. There are many advantages in doing so, primary amongst which is speed, but unfortunately, getting the GPU to handle tasks traditionally performed by the CPU isn't quite so simple .CUDA was developed by NVIDIA to execute simple programs using GPGPU which were executed on CPU. The logic behind the idea is that GPU consists of multi core processing units which operate in parallel and can be used to execute multiple instructions concurrently. CUDA gives program developers direct access to the virtual instruction set and memory of parallel computation elements in CUDA GPU's.

KEYWORDS: Parallel Programming, Graphic processing Unit (GPU), Compute Unified Device Architecture (CUDA), Multi Cores.

1. INTRODUCTION

[1] In recent years the application of graphics processing units to general purpose computing has become widely developed. Parallel programming is becoming one of the hottest topics in software today as multi-core CPUs decrease in price and increase in power. Parallelism in programs allows multiple processes to be executed concurrently using separate threads and processing units. This is appealing to developers and users alike, because it can help reduce runtimes while still producing the same results as if it were run in serial.

To give an example, let's say we have an array(Fig 1.1.) that contains thousands of integer elements and each value needs to be processed through a lengthy algorithm. Instead of running each value through the algorithm consecutively (i.e. one at a time), parallelism allows multiple values to be processed simultaneously (i.e. running many values through the algorithm at the same time), reducing overall processing time and producing fast and accurate

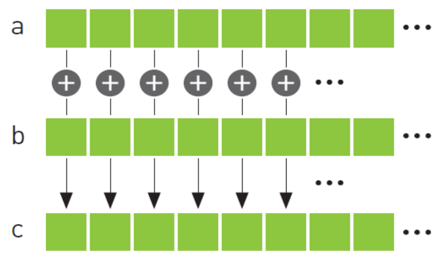


Fig 1.1 results

The aim of this paper is to greatly reduce the high computing time of the CPU. GPUs are characterized by numerous simple yet energy-efficient computational cores, thousands of simultaneously active fine-grained threads, and large off chip memory bandwidth. While CPU is made of multiple cores, GPU is made of thousands of cores. CPU + GPU is a

ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

powerful combination because CPUs consist of a few cores optimized for serial processing, while GPUs consist of thousands of smaller, more efficient cores designed for parallel performance.(Fig 1.2) Serial portions of the code run on the CPU while parallel portions run on the GPU.

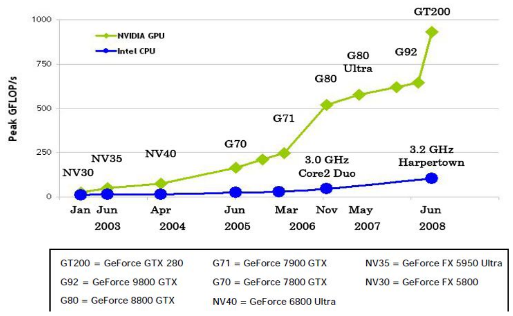


Fig 1.2

The rest of this paper is organized as follows. Section 2 reviews the comparison between CPU and GPU. Section 3 tells about the flow of process in CUDA. Section 4 deals with working on CUDA. Applications using GPU are listed in Section 5. Finally, Section 6 concludes and raises present scenario of GPU.

II. COMPARISON BETWEEN CPU AND GPU

The difference between CPUs and GPUs is that GPUs are highly specialized in number crunching, something that graphics processing desperately needs as it involves millions, if not billions, of calculations per second. The amount of cores that GPUs have depends on the manufacturer. NVidia graphics solutions tend to pack more power into fewer chips, while AMD solutions pack in more cores to increase processing power. Typical high-end graphics cards have 68 cores if it’s nVidia, and ~1500 cores if it’s AMD. Architecturally, the CPU is composed of a only few cores with lots of cache memory that can handle a few software threads at a time. In contrast, a GPU is composed of hundreds of cores that can handle thousands of threads simultaneously. The ability of a GPU with 100+ cores to process thousands of threads can accelerate some software by 100x over a CPU alone (Fig 2.1). What’s more, the GPU achieves this acceleration while being more power- and cost-efficient than a CPU.

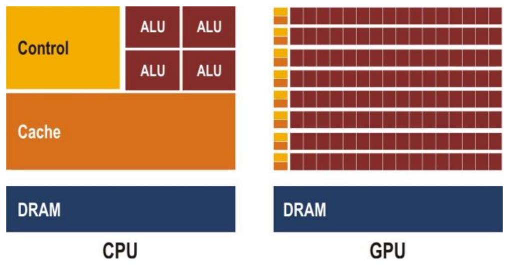


Fig 2.1
GPU devotes much more transistors to data processing rather than data caching and flow control.

III.FLOW OF PROCESS IN CUDA

[1] CUDA is an extension to C based on a few easily-learned abstractions for parallel programming and a few corresponding additions to C syntax. CUDA represents the coprocessor as a device that can run a large number of threads. The threads are managed by representing parallel tasks as kernels mapped over a domain. Data is prepared for processing on the GPU by copying it to the graphics board's memory. Data transfer is performed using DMA and can take place concurrently with kernel processing. CUDA gives program developers direct access to the virtual instruction set and memory of the parallel computational elements in CUDA GPUs.

ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

3.1 Data transfer from main memory to device memory.

Consider Fig 3.1.The input data for parallel proceesing are got from the user using CPU. Memory is allocated for the data in both CPU and GPU. After getting all data, the data is copies from the main memory(CPU) to the device memory(GPU).The CPU instructs the GPU for parallel processing [2].

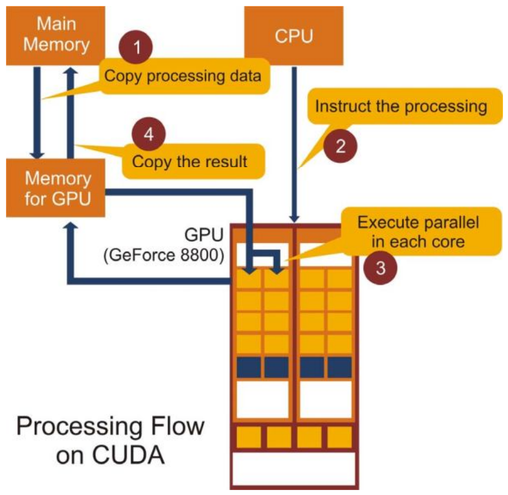


FIG 3.1

1. Copy data from main mem to GPU mem
2. CPU instructs the process to GPU
3. GPU execute parallel in each core
4. Copy the result from GPU mem to main mem

3.2 Data Transfer between Host and Device

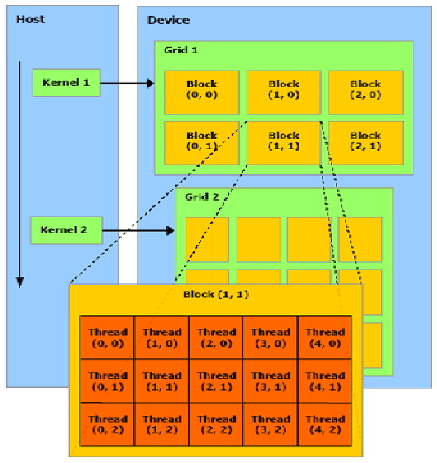
The bandwidth between device memory and the device is much higher than the bandwidth between device memory and host memory. Therefore, one should strive to minimize data transfer between the host and the device, for example, by moving more code from the host to the device(Fig 3.2), even if that means running kernels with low parallelism computations. Intermediate data structures may be created in device memory, operated on by the device, and destroyed without ever being mapped by the host or copied to host memory. Also, because of the overhead associated with each transfer, batching many small transfers into a big one always performs much better than making each transfer separately. Finally, higher performance for data transfers between host and device is achieved by using page-locked host memory. In addition, when using mapped page-locked memory there is no need to allocate any device memory and to explicitly copy data between device and host memory. Data transfers are implicitly performed each time the kernel accesses the mapped memory. For maximum performance, these memory accesses must be coalesced like if they were accesses to global memory. Assuming that they are and that the mapped memory is read or written only once, using mapped page-locked memory instead of explicit copies between device and host memory can be a win performance-wise. On integrated systems where device memory and host memory are physically the same, any copy between host and device memory is superfluous and mapped page locked memory should be used instead.

ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014



The host issues a succession of kernel invocations to the device. Each kernel is executed as a batch of threads organized as a grid of thread blocks

Fig 3.2

IV. WORKING ON CUDA

[1] Nvidia CUDA Programming Basics consists of The Programming model, The Memory model and CUDA API basics. In the CUDA Programming Model the GPU is seen as a compute device to execute a portion of an application, a function for example, that has to be executed many times, can be isolated as a function, works independently on different data. Such a function can be compiled to run on the device. The resulting program is called a Kernel.(Fig 4.1). The batch of threads that executes a kernel is organized as a grid of thread blocks. In the Memory model each CUDA device has several memories that can be used by programmers to achieve high Computation to Global Memory Access (CGMA) ratio and thus high execution speed in their kernels. Variables that reside in registers and shared memories can be accessed at very high speed in a highly parallel manner. Registers are allocated to individual threads; each thread can only access its own registers. A kernel function typically uses registers to hold frequently accessed variables that are private to each thread. Shared memories are allocated to thread blocks;

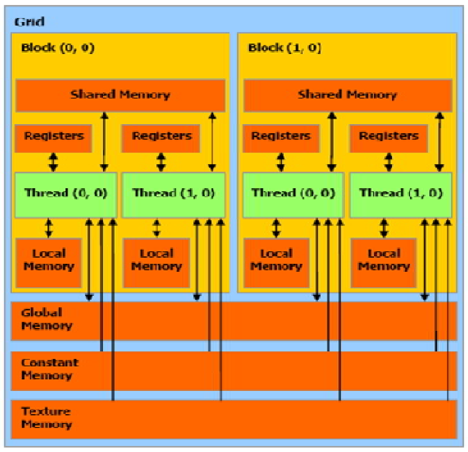


Fig 4.1

each thread can only access its own registers. A kernel function typically uses registers to hold frequently accessed variables that are private to each thread. Shared memories are allocated to thread blocks; all threads in a block can

ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

access variables in the shared memory locations allocated to the block. Shared memories are efficient means for threads to cooperate by sharing the results of their work. At the middle of the table, we see global memory constant memory. These are the memories that the host code can write (W) and read (R) by calling API functions. The global memory can be accessed by all the threads at anytime of program execution. The constant memory allows read-only access by the device and provides faster and more parallel data access paths for CUDA kernel execution than the global memory.

CUDA defines registers, shared memory, and constant memory that can be accessed at higher speed and in a more parallel manner than the global memory. Using these memories effectively will likely require re-design of the algorithm. It is important for CUDA programmers to be aware of the limited sizes of these special memories. Their capacities are implementation dependent. Once their capacities are exceeded, they become limiting factors for the number of threads that can be assigned to each shared memory.

V. APPLICATIONS USING GPU

A wide variety of applications have achieved dramatic speedups with GPGPU implementations.

[3] A new guideline for the design and implementation of effective LSMs (Local search metaheuristics) on GPU is proposed. Very efficient approaches are proposed for CPU-GPU data transfer optimization, thread control, mapping of neighboring solutions to GPU threads, and memory management. These approaches have been experimented using four well-known combinatorial and continuous optimization problems and four GPU configurations. Compared to a CPU-based execution, accelerations up to X80 are reported for the large combinatorial problems and up to X240 for a continuous problem. Finally, extensive experiments demonstrate the strong potential of GPU-based LSMs compared to cluster or grid-based parallel architectures.

[4] Nvidia CUDA framework for bitmap based association rule mining algorithm is proposed and its performance statistics on CPU and GPU is studied. It can solve complex financial problems from 4X to 20X faster than solving same problems on CPU based implementations As the financial industry is growing exponentially there is an extreme need of molding the huge amount of data into strategic information in a very short span of time. Using these less expensive GPUs even small organizations can build highly computational intensive environment to solve day to day complex computations in order to take competitive advantage which is critically required to survive.

[5] The goal of the paper is to investigate if the GPUs can be useful accelerators for BI analytics with very large data sets that cannot fit into GPU's onboard memory. GPU-accelerated implementations of analytics potentially provide better raw performance, better cost-performance ratios, and better energy performance ratios.

[6] The data sets produced in radiological exams are growing larger everyday, which creates a severe demand on computing power for image segmentation algorithms. To address such demand, this work as introduced a CUDA implementation of a widely used fuzzy connected image segmentation method on low-cost GPUs. The results show that the CUDA implementation achieves a speedup from 7.2x to 14.4x over an optimized CPU implementation.

[7] The paper has shown that PSO, provided with the right fitness function, can be effective in detecting traffic signs in real time. To speed up execution times, the algorithm exploits the parallelism offered by modern graphics cards and, in particular, the CUDA architecture by NVIDIA. The effectiveness of the approach has been assessed on a synthetic video sequence, which has been successfully processed in real time at full frame rate.

[8] The paper has demonstrated how clustering for visualization of large line data can be done efficiently with a combination of multiple GPUs and CPUs. This parallel clustering method employs the Expectation Maximization algorithm to iteratively approximate the optimal data partitioning.

[9] In order to take advantage of high number of cores, the paper presents a clustering scheme and collision-packet traversal to perform efficient collision queries on multiple configurations simultaneously. They have implemented our algorithms on commodity NVIDIA GPUs using CUDA and can perform 500, 000 collision queries/second on our benchmarks, which is 10X faster than prior GPU-based techniques.

[10] A framework for solving linear algebra problems on graphics processors is presented.

ISSN(Online):2320-9801
ISSN(Print): 2320- 9798

International Journal of Innovative Research in Computer and Communication Engineering

(An ISO 3297: 2007 Certified Organization)

Vol. 2, Special Issue 3, July 2014

[11] Harris et al. present a cloud dynamics simulation using partial differential equations

[12] The molecular dynamics simulations have also shown impressive speedups using GPU's.

[13] The pixel engines using GPU are used in database operations for speedup.

[14] M. Schatz et al uses GPU in sequence alignment and

[15] AES encryption algorithms have been successfully implemented on GPUs.

VI. CONCLUSION

When we look at a typical computer chip, a CPU chip, versus a GPU chip, a GPU chip tends to give us 10 times more peak execution throughput and around six times of memory DRAM access bandwidth. With the announcement of a new Blue Waters petascale system that includes a considerable amount of GPU capability, it is clear GPUs are the future of supercomputing. The CUDA platform is a foundation that supports a diverse parallel computing ecosystem.

REFERENCES

[1] NVIDIA, CUDA programming guide 2.3,http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_2.3.pdf.
[2] CUDA Wikipedia reference
[3] Van Luong, Nouredine Melab, and El-Ghazali Talbi, GPU Computing for Parallel Local Search Metaheuristic Algorithms, IEEE TRANSACTIONS ON COMPUTERS, VOL. 62, NO. 1, JANUARY 2013.
[4] Adil, S.H, Qamar, S., Implementation of association rule mining using CUDA Emerging Technologies, 2009. ICET 2009. International Conference on Page(s): 332 - 336 E-ISBN : 978-1-4244-5631-4
[5] Ren Wu, Bin Zhang, Meichun Hsu, GPU-Accelerated Large Scale Analytics, HP Labs
[6] Ying Zhuge, Yong Cao, Miller, R.W., GPU accelerated fuzzy connected image segmentation by using CUDA, Engineering in Medicine and Biology Society, 2009. EMBC 2009. Annual International Conference of the IEEE Page(s):6341 - 6344 ISSN : 1557-170X E-ISBN : 978-1-4244-3296-7
[7] Mussi, L. , Cagnoni, S, Daolio, F. GPU-Based Road Sign Detection Using Particle Swarm Optimization, Intelligent Systems Design and Applications, 2009. ISDA '09. Ninth International Conference .Page(s): 152 - 157 E-ISBN : 978-0-7695-3872-3
[8] Jishang Wei, Hongfeng Yu, Chen, J.H., Kwan-Liu Ma, Parallel clustering for visualizing large scientific line data, Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium Page(s): 47 - 55 Print ISBN: 978-1-4673-0156-5
[9] Jia Pan and Dinesh Manocha, GPU-based Parallel Collision Detection for Real-Time Motion Planning, Journal International Journal of Robotics Research Volume 31 Issue 2, February 2012 Pages 187-200.
[10] J. Kr¨uger, R. Westermann, Linear algebra operators for GPU implementation of numerical algorithms, ACM Transactions on Graphics 22 (3) (2003) 908.916.
[11] M. J. Harris, W. V. Baxter, T. Scheuermann, A. Lastra, Simulation of cloud dynamics on graphics hardware, in: Proc. of the ACM SIGGRAPH/EUROGRAPHICS
Conference on Graphics Hardware, 2003.
[12] C. I. Rodrigues, D. J. Hardy, J. E. Stone, K. Schulten, W.-M. W. Hwu, GPU acceleration of cutoff pair potentials for molecular modeling applications, in: Proc. of the 2008 Conference on Computing Frontiers, 2008.
[13] N. K. Govindaraju, B. Lloyd, W. Wang, M. Lin, D. Manocha, Fast computation of database operations using graphics processors, in: Proc. of the ACM SIGMOD
International Conference on Management of Data, 2004.
[14] M. Schatz, C. Trapnell, A. Delcher, A. Varshney, High-throughput sequence alignment using Graphics Processing Units, BMC Bioinformatics 8 (1) (2007) 474.
[15] T. Yamanouchi, AES encryption and decryption on the GPU, GPU Gems 3, Addison Wesley, 2007, pp. 785-803.