Comparison Study of Parallel Computing with ALU and GPU (CUDA)

Joe Johnson[1], G. Vijayalakshmi[2]

[1]Computer Science and Engineering, SRM University, Chennai, Kattankulathur, Tamilnadu, India

[2]Assistant Professor, Computer Science and Engineering, SRM University, Chennai, Kattankulathur, Tamilnadu, India

Abstract: ***Computers with high performance and speed is taking over almost all the activities in the modern world of today, reducing human effort hence forth Parallelization is a feature that is exploited to provide better performance computing these days. During the execution of a C program, only sequential execution takes place resulting in wastage of time &other valuable system resources. This project results in parallelizing the C program such that it is executed as parallel threads by all the cores of the processor or in the case of GPU executing as threads which are passed to around 200 cores. NVIDIA's CUDA architecture provides a powerful platform for writing highly parallel programs executed by GPU. This project is a comparison study between both ALU &GPU (CUDA) parallelization which results in high performance parallel computing with much more efficiency than the conventional processing.***

Keywords: CUDA, Hyper Threading, Parallel Computing**.**

1. Introduction

Introduction- 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 ("in parallel"). There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Parallelism has been employed for many years, mainly in high-performance computing

Parallelization is a feature that is exploited to provide better performance computing these days. During the execution of a C program, only sequential execution takes place resulting in wastage of time &other valuable system resources. This project results in parallelizing the C program such that it is executed as parallel threads by all the cores of the processor or in the case of GPU executing as threads which are passed to around 200 cores. NVIDIA's CUDA architecture provides a powerful platform for writing highly parallel programs executed by GPU. This project is a comparison study between both ALU &GPU (CUDA) parallelization which results in high performance parallel computing with much more efficiency than the conventional processing.

Some of the major limitation constraint for the Smart phones is as follows:

1. Execution Time constraint
2. CPU Cycle constraint
3. Memory constraint
4. GPU Cycle constraint

In further sections we are going to describe methods by which parallelism can be exploited.

2. ALU Parallel Execution

While execution of c/c++ program even though processor might have more than 1 core (dual core=2 cores, quad core=4 cores), it will presume execution sequentially resulting in wastage of GPU time and other valuable resources. This project deals with exploiting the parallel elements in the program and executing it in a parallel manner. Here we find out the dependency's present in the program, mainly in the variable being used using different algorithms, we insert tags or directives stating which all variables or lines can be executed in parallel and the rest which should be executed sequentially. So if while executing in a quad core machine 4 threads can be processed simultaneously. Thus decreasing the execution time to quarter of the real execution time and increasing the usage of resources.

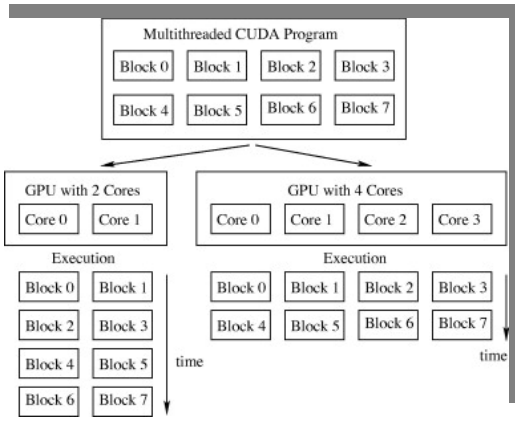


Figure 1: Distributed computing

3. GPU Parallel Execution

CUDA is a parallel computing platform and programming model created by NVIDIA.NVIDIA graphics processing units (GPUs) in fig1.3 implement the CUDA architecture and programming model. The CUDA platform is accessible to software developers through CUDA-accelerated libraries, compiler directives (such as OpenACC), and extensions to industry-standard programming languages, including C, C++ and Fortran. C/C++ programmers use 'CUDA C/C++' (C/C++ with CUDA extensions to express parallelism, data locality, and thread cooperation).CUDA is a plat form

dependant architecture, CUDA architecture can be implemented along with the OpenMP API which includes tags to the program thus making it a parallel executable by the GPU.

Compute capability (version)	GPUs	Cards
1.0	G80, G92, G92b, G94, G94b	GeForce 8800GTX/Ultra, 9400GT, 9600GT, 9800GT, Tesla C/D/S870, FX4/5600, 360M, GT 420
1.1	G86, G84, G98, G96, G96b, G94, G94b, G92, G92b	GeForce 8400GS/GT, 8600GT/GTS, 8800GT/GTS, 9600 GSO, 9800GTX/GX2, GTS 250, GT 120/30/40, FX 4/570, 3/580, 17/18/3700, 4700x2, 1xxM, 32/370M, 3/5/770M, 16/17/27/28/36/37/3800M, NVS420/50
1.2	GT218, GT216, GT215	GeForce 210, GT 220/40, FX380 LP, 1800M, 370/380M, NVS 2/3100M
1.3	GT200, GT200b	GeForce GTX 260, GTX 275, GTX 280, GTX 285, GTX 295, Tesla C/M1060, S1070, Quadro CX, FX 3/4/5800
2.0	GF100, GF110	GeForce (GF100) GTX 465, GTX 470, GTX 480, Tesla C2050, C2070, S/M2050/70, Quadro Plex 7000, Quadro 4000, 5000, 6000, GeForce (GF110) GTX 560 TI 448, GTX570, GTX580, GTX590
2.1	GF104, GF114, GF116, GF108, GF106	GeForce GT 430, GT 440, GTS 450, GTX 460, GTX 550 Ti, GTX 560, GTX 560 Ti, 500M, Quadro 600, 2000
3.0	GK104, GK106, GK107	GeForce GTX 690, GTX 680, GTX 670, GTX 660 Ti, GTX 660, GTX 650, GT 640, GeForce GTX 680M, GTX 660M, GeForce GT 650M, GeForce GT 640M
3.5	GK110	

Figure 1.2: Compute capability table (version of CUDA supported) by GPU and card. Also available directly from Nvidia

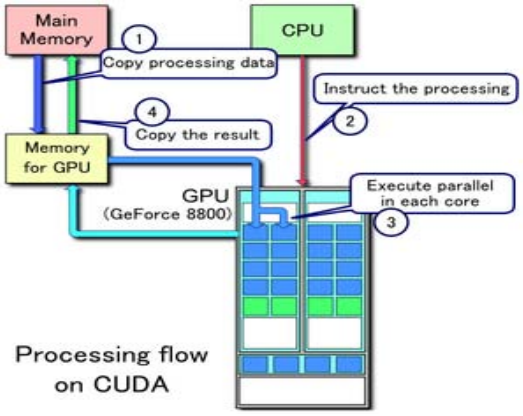


Figure 1.3: CUDA Processing Flow

Objective of the project is to provide high performance computing using exploiting parallelism of ALU and GPU computing. This project is implemented using light weight threads. Threads use atomic operations in a global memory space shared by all threads. This project is to implement automatic parallelizing of C program for both ALU and GPU (CUDA) execution. This enables a comparison study between ALU parallel processing and GPU parallel processing with CUDA implemented with the help of OpenMP API. This comparison study helps to check in to the improvement in parameters which is associated with parallel computing like

- Execution Time
- RAM Usage
- Memory Usage
- Cycle Usage

4. Proposed System Architecture and Implementation Method Details

ALU parallel processing and GPU parallel processing were considered as two ends of a line but, it is possible to combine them in such a way that high parallelism can be achieved in execution phase. For this a synchronous performance between the ALU and GPU is expected. During the implementation of ALU parallel execution the program is being Re-Engineered using parallel concepts so that during execution it can be divided in to treads and executed by powerful multiple cores of the CPU.

The components are as follows

- Host(ALU)
- Device(NVIDIA GPU)

The main constraints to be considered when implementing the CUDA are

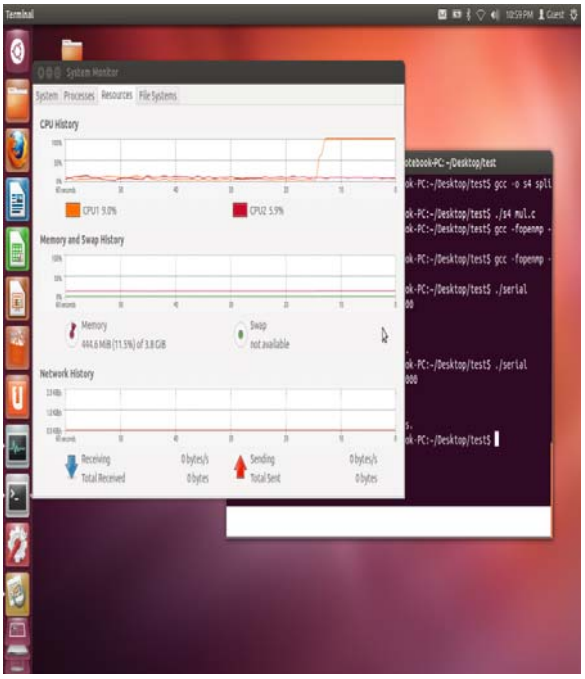
- Data transfer constraint
- Processing constraint
- Result write back constraint

When CUDA execution is initiated the address and data to be processed are transferred from the host (ALU) to the device (GPU).The next step is the execution in which threads are processed as blocks by the GPU cores. Last step is to write back the result to the host (ALU) and display the output.

5. Results

First frame shows the System Monitor when the program is being executed in Serial mode. We can observe that in this Dual core machine only single core is used at a time Which results in

- More Execution Time
- Wastage of CPU Cycle
- Resource under utilization

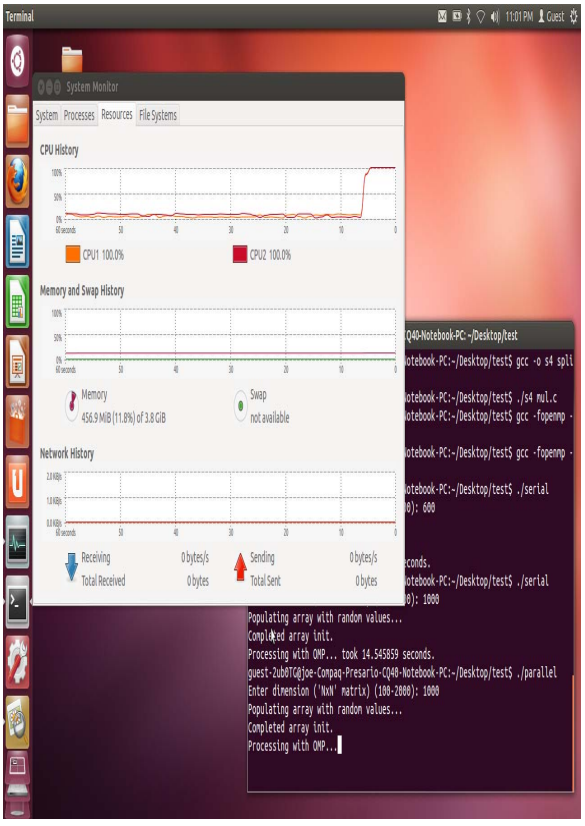


Next frame shows the System Monitor when the program is being executed in parallel. We can observe that in this Dual core machine both the cores are being used up at the same time decreasing the

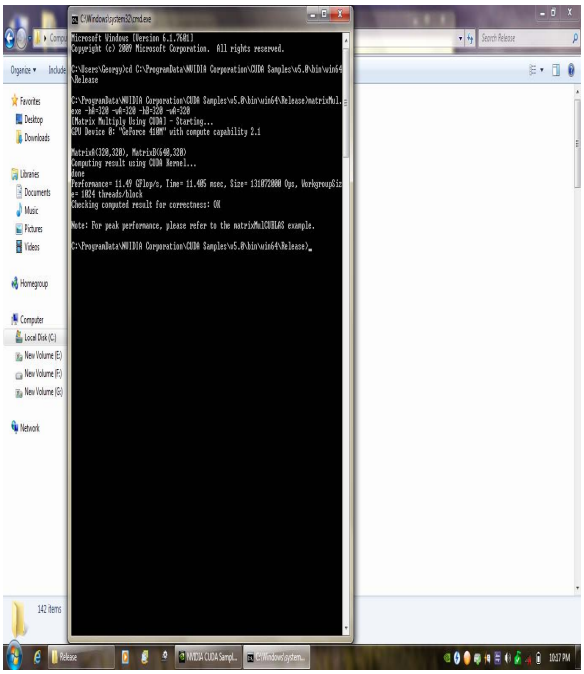
- Execution Time
- Delays

While the factors increased are

- CPU utilization
- Speed of execution
- Cycle usage



Next frame shows the program being executed in parallel using CUDA. We can observe that in this Dual core machine with NVIDIA GPU is processing the results with much more efficiency and less time

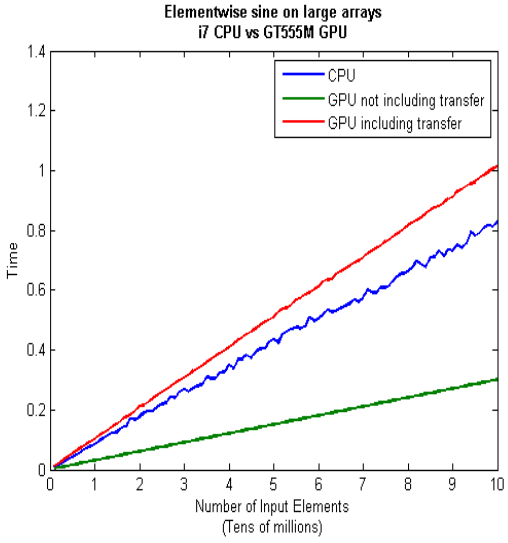


6. Conclusion

The graph shows the results plotted when the program is being executed in Serial mode, Parallel mode and CUDA mode. The resulting graph shows that the following factors are affected by the implementation.

- Execution Time
- Delays
- CPU utilization
- Speed of Execution
- Transfer Time

We can conclude that the results can be more efficiently got when GPU Execution is Implemented comparing to normal ALU execution



References

[1] J. Nickolls et al., ‘‘Scalable Parallel Programming with CUDA,’’ ACM Queue,vol. 6, no. 2, Mar./Apr. 2008, pp. 40-53.

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

[3] NVIDIA, NVIDIA CUDA Programming Guide,2009; http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf

[4] Modern Compilers: Theory and Practice, Shankar Balachandran, Dept. of CSE, IIT Madras

[5] W.R. Mark et al., ‘‘Cg: A System for Programming Graphics Hardware in a C-like Language,’’ Proc. Special Interest Group on Computer Graphics (Siggraph), ACM Press, 2003, pp. 896-907.

[6] X. Tian, A. Bik, M. Girkar, P. Grey, H. Saito, and E. Su, “Intel OpenMP C++/Fortran Compiler for Hyper-Threading Technology: Implementation and Performance”, Intel Technology Journal, Q1, 2002. (http://www.intel.com/technology/itj)

[7] NVIDIA, ‘‘NVIDIA’s Next Generation CUDA Compute Architecture,’’ 2009; http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Compute_Architecture_Whitepaper.pdf.

[8] 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.

Author Profile

Joe Johnson received the B.Tech. and M.Tech degrees in Computer Engineering from Rajagiri School of Engineering Technology and SRM University in 2011 and 2013, respectively.