Implementation of High Speed Convolution Algorithm on CUDA based Graphics Processing Unit

G. Prasad Acharya
Department of Electronics and Communication Engineering, Sreenidhi Institute of Science and Technology, Hyderabad, TS, India

N. Srinivasa Reddy
Department of Electronics and Communication Engineering, Sreenidhi Institute of Science and Technology, Hyderabad, TS, India

S.P.V. Subba Rao
Department of Electronics and Communication Engineering, Sreenidhi Institute of Science and Technology, Hyderabad, TS, India

ABSTRACT

The advancements in multi processors based computers with parallel computing has increased the computational speed . The multi processors consists of hundreds of processor cores or graphics processing units are designed for multimedia applications to improve the pixel resolution .These processors are also used for general computations are called as General Processing GPU (GP-GPU) . The exploration of multi cores in CUDA (Compute Unified Device Architecture) led to parallel computation . CUDA C is a high level programming language released by NVIDIA in 2006 for its NVIDIA GPUs. In this paper a high speed convolution algorithm is implemented on CUDA based graphics processing unit. The implemented algorithm is evaluated based on computational speed. Simulation results shows that computational speed by GPU has been increased by many folds when compared with CPU.

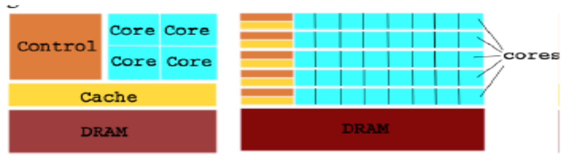
Keywords

CUDA, GP-GPU,DFT,Convolution, CUDA C

1. INTRODUCTION

The single core processor has the limitations of limited performance speed for large computations in real time multimedia applications. Multi core CPUS are used to increase the speed as depicted in Figure 1. Though Multi core processors are capable of executing (MIPS) Millions Instructions per Second on a large database, there are some computational problems that still remain to be addressed by the processor designers. Building high performing multi processors require an expensive and intense design cycle. Computations with the more powerful microprocessor for processing real time multimedia applications, Internet of Things (IoT), Cloud and Mobile Computing, Big Data Analytics requires more computational speed as complex algorithms and huge database is involved. The programmers in the past have developed a different concept called parallel processing. Parallel processing involves two or more processors and a task is divided into number of subtasks by a special function of the operating system and processed by a processor.

These sub-tasks are then scheduled for execution into a number of cores for parallel execution. Each core then executes its part of total task. After the completion of execution of individual tasks, the Operating System reassembles the final result by integrating the individual sub-task result.



(a) (b)

Figure 1: (a) CPU (b) GPU

The computational speed, complexity and power consumption of these CPU-based parallel processors are limited by the number of processor cores that can reside on a single processor.For floating point operations CPU based multi-core processors lag behind the GPUs in terms of speed . This is because CPUs has large portion of die space for control logic circuitry and cache memory, whereas GPU allocates large die space to accommodate large number of floating-point Arithmetic and Logic Units called cores. This architectural feature in GPUs enables splitting of a task into a number of threads. These threads are then executed in parallel by scheduling each thread into an independent core and hence improving the performance speed.

Many of signal processing applications commonly used by engineers and scientists need a specialized parallel architecture as available in GPUs in order to carry out specialized algorithms. These algorithms include Linear and Circular Convolutions, Multiply-And-Accumulate (MAC) operations, Discrete Fourier transform (DFT). The parallel architectures available in GPUs are best suited for this type of algorithms, as the same operation is applied independently to different modules of data and software development tools. This simplifies the process of parallelizing code without the need for additional work by programmer.

The DFT and Convolution are the most sought after algorithms in signal and image processing applications, which require a large number of MAC operations on a large volume of sampled data. In this paper, the implementation of DFT and convolution algorithms using a parallel architecture called CUDA, which was conceptualized by NVIDIA, is presented.

2. BACKGROUND OF DFT AND CONVOLUTION

2.1 Discrete Fourier Transform (DFT)

The Discrete Fourier transform is very powerful computational basis for the spectral analysis of discrete data in frequency domain. Spectral analysis is the process of identifying various frequency components present in data along with their amplitude and phase values .The DFT transforms time or space domain data into frequency domain data. The DFT is one of the most important mathematical tools in Digital Signal Processing that provides a basis for computing on digital computer or special purpose processor. The DFT has wide spread applications in determining system frequency response, speech, image and video processing areas and so on. The N-point DFT of a time domain signal x(n) is denoted by X (K) which is periodic with a period of N. The N- point DFT X(K) is obtained by finding N equally spaced samples of Fourier transform of the discrete signal x(n) over a period of 2π. In addition to determining the frequency content of a signal, DFT is also useful in performing linear filtering operations in the frequency domain. The Discrete Fourier Transform is a numerical variant of the Fourier Transform. Specifically, for a given discrete data of n input amplitudes such as {x0, x1, x2... xn-2, xn-1}, the Discrete Fourier Transform yields a set of n frequency components.

The DFT of a signal x (n) is defined as:

$$X[K] = \sum_{n=0}^{N-1} x(n)\ W_N^{kn}\ , \quad k= 0, 1, ---, N-1.$$

Where $W_N = e^{\frac{-j2\pi}{N}}$

The inverse discrete flourier transform (IDFT), which recovers back the original signal x(n) in time domain

$x[n] = (1/N)\sum_{n=0}^{N-1} X(K)\ W_N^{-kn}$, for n= 0, 1,--, N-1.

Here K denotes the frequency domain ordinal, and n represents the time-domain ordinal. The "N" is the length of the DFT.

2.2 Convolution

Convolution is another powerful important mathematical tool useful in finding the response of a Linear-Time Invariant (LTI) system. It has many applications in various research areas like telecommunications, optical communications, Bio-medical imaging, speech and image processing, electrical engineering, computer graphics and radio astronomy etc. The huge amount of data involved in all these applications possesses a big challenge in data storage, classifying, processing and retrieving of information [1]. All thee challenges reduce the execution speed of the convolution. The execution speed can be improved drastically by performing the MAC operations on individual parts of the data on the multiple parallel-cores available in GPUs. GPUs have many small processing elements, called cores and, therefore, are well-suited for computing MAC operations in convolution.

Convolution of two sequences x (n) and h(n) is defined as:

$$y[n] = \sum\ x(l).h(n-l) \qquad (4)$$

Where, x (n) is a discrete-time input signal, h(n) is an impulse response of an LTI system, y(n) is the output or response of LTI system. The convolution summation can be easily implemented using a direct form of Finite Impulse Response (FIR) filter. Convolving in the time domain mentioned in above equation has no inherent latency at the output y(n). However, the computational cost of convolution depends on the number of multiply-add operations per output sample. It can easily be verified that with increasing the length of the filter coefficients results in increase in the computational cost. Consequently, this direct computation method cannot be best suited in evaluating convolutions in real time for long sequences. Some efficient methods are available in frequency domain when compared to that in time domain. By using Discrete Time Fourier Transform (DTFT) of a discrete sequence, a signal can be converted into corresponding frequency domain. The DTFT of a discrete sequence is a continuous periodic function of frequency. In general, the sequence x(n) is of finite duration sequence and the DTFT of it being continuous, can be sampled at uniformly spaced intervals over a period of 2π with sampling interval of $2\pi/N$. This sampling enables a new type of transformation technique known as N point Discrete Fourier Transform (DFT). The linear convolution of two signals or sequences in time domain leads to multiplication of their DTFTs. But in the case of DFT domain, Inverse DFT (IDFT) of multiplication of the DFTs of two sequences does not produce linear convolution but results in circular convolution of the two sequences. This circular convolution is denoted by, X (k) and H (k), which are the N-point DFTs of x(n) and h(n), respectively. Thus, the inverse DFT of product of X (k) and H (k) is not useful to find the linear convolution of x(n) with h(n), which means IDFT{X(k) H (k)}≠x(n) * h(n). However, the linear convolution of two sequences can be computed by using DFT and IDFT if the two sequences are properly padded with zeros.

3. CUDA ARCHITECTURE

CUDA, shown in Figure 2 provides a parallel computing platform wherein the task assigned to a processing unit is divided into a number of threads and then executed in parallel among the multiple-cores available. The programming model using CUDA C had been invented by NVIDIA, which enables drastic increases in computing speed. With CUDA-enabled GPUs, software developers, scientists and researchers are finding broad-ranging uses for GPU computing with CUDA. The convolution algorithm is one of the powerful and most commonly used algorithms to characterize and obtain the frequency spectrum of the given system that applies to a set of data samples. The CUDA provides a computing platform for intensive and big volume of data, we proposed the Graphics Processing Units (GPU) based Convolution algorithm which can be the cost effective solution. The GPU can execute the Convolution application in parallel using the mode of Single Instruction Multiple Data (SIMD). GPU is now becoming a hot research topic that works in image processing, video and multimedia applications. For this reason, it is chosen to develop the compute and to reduce the Convolution and DFT calculation time. To execute this transform on GPU, we used a specific architecture that's CUDA.

A. Programming with CUDA

The CUDA computing engine virtualizes graphics hardware available to the programmer through the use of uniquely numbered threads that are organized into 1D, 2D, or 3D blocks of arbitrary size.

B. Processing flow of CUDA
Figure 3 illustrates the Processing flow of CUDA.

Step 1. Copy the data set for processing from CPU memory to GPU memory.

Step 2. CPU instructs the GPU to perform the given operation in the form of Kernel.

Step 3. Load GPU program and execute

Step 4. Copy and store the results from GPU memory to CPU memory.

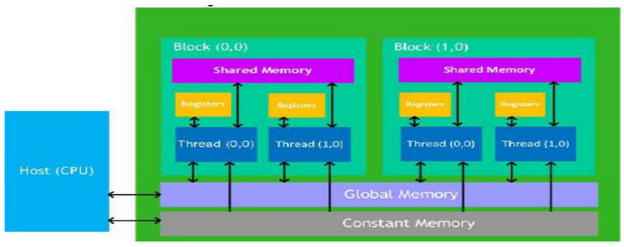


Figure 2: GPU Device memory Hierarchy

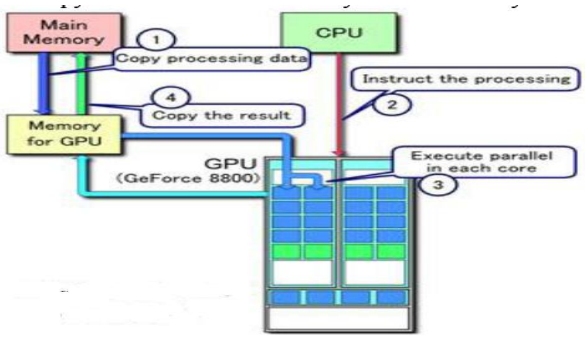


Figure 3: Processing flow of CUDA

C. Programming Model

CUDA C is the language that supports heterogeneous computing on a GPU platform. The CUDA environment consists of both host device and target device. A CUDA program is a unified source code on the host, which consists of both host code and device code. The CPU acts as a host, fetching and decoding the instructions and exhibits little or no parallelism. The CPU can schedule the device code (instruction/program) to get executed in the target GPU which supports parallel programming. NVIDIA C compiler separates host code and device code during the compilation process. The host code is a simple C code, which is compiled further by regular C compilers, and the device code is NVIDIA C extensions, for device parallelism, which can be launched by special device functions called "kernels". The device code further compiled by NVCC (NVIDIA C Compiler) and executed on the GPU (device).

The CUDA program calls parallel kernels executing a set of parallel threads. The CUDA programming model, as shown in Figure 4 consists of thread blocks and grids of thread blocks. A group of threads is called Thread Block and a group of such Thread Blocks is known as a Grid. The threads in a thread block are aligned in one-dimensional, two-dimensional, or three-dimensional form. In a similar way, thread blocks in a grid are aligned in a one-dimensional, two-dimensional or three-dimensional form. This provides a natural way to invoke computations across the elements in a domain such as a vector, matrix, or volume. The programmer or compiler organizes these threads in the number of specified thread blocks and grids of thread blocks.

The kernel can be invoked indicating the number of threads by using the execution syntax shown below:

Kernel Name <<< Number of Threads >>> Parameters

Each thread has a unique "thread index", which can be accessible using "threadIdx".

The thread ID can be calculated based on its index of thread. For a one dimensional block they are the same. For a two dimensional block of size (Dx,Dy), if the thread index is (x, y) then its thread ID is $(x+y\cdot Dx)$. For a three dimensional block of size (Dx,Dy,Dz), if the thread index is (x, y, z) then its thread ID is $(x + y \cdot Dx + z \cdot Dx \cdot Dy)$.

The number of blocks per grid (nB) and the number of threads per block (nT) are specified in a kernel launch as

Kernel name <<< nB, nT >>> (parameters)

Where nB = (4,2), represents a two dimensional blocks by 4 columns and 2 rows and a total of $4 \cdot 2 = 8$ blocks. Each block has 10 numbers of threads. Each thread in a thread block and each block in a grid has unique thread index and block index.

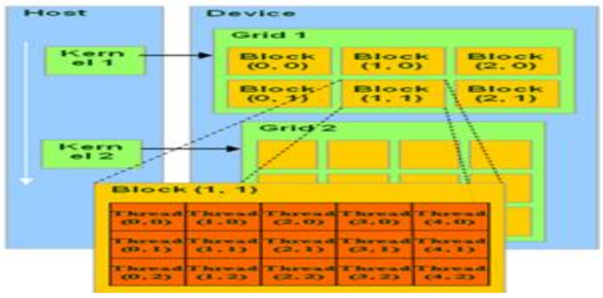


Figure 4: CUDA Programming Model

4. IMPLEMENTATION OF CONVOLUTION ON GPU

As discussed in section 2, the convolution of two signals x(n) and h(n) shall be calculated in three steps

Step 1: Make the lengths of both x(n) and h(n) equal to N= l +m-1 by suitable zero padding , where l ,m being lengths of x(n) and h(n) respectively.

Step 2: Calculate N-Point DFT of both the sequences, i.e., X(K) and H(K).

Step 3: Evaluate N-point DFT Y(K) by multiplying X(K) and H(K).

Step 4: Compute N-Point IDFT of Y(K) to obtain convolved sequence y(n)

The procedure to implement the convolution algorithm on GP-GPU is depicted in the flowchart shown in Figure 5.

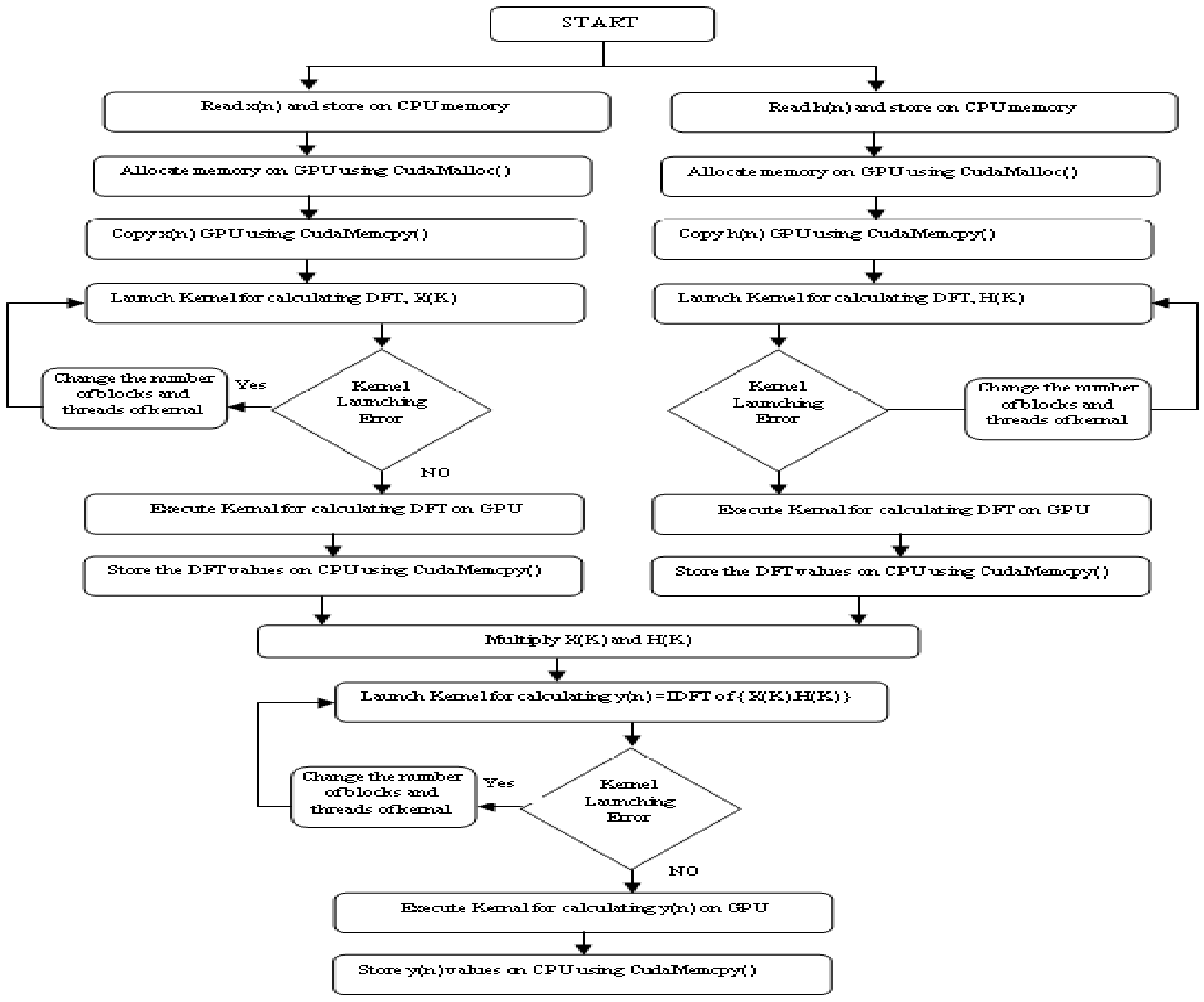

START
Read x(n) and store on CPU memory
Read h(n) and store on CPU memory
Allocate memory on GPU using CudaMalloc()
Copy x(n) GPU using CudaMemcpy()
Copy h(n) GPU using CudaMemcpy()
Launch Kernel for calculating DFT, X(K)
Launch Kernel for calculating DFT, H(K)
Change the number of blocks and threads of kernal
Yes
Kernel Launching Error
NO
Execute Kernal for calculating DFT on GPU
Store the DFT values on CPU using CudaMemcpy()
Multiply X(K) and H(K)
Launch Kernel for calculating y(n) = IDFT of (X(K).H(K))
Execute Kernal for calculating y(n) on GPU
Store y(n) values on CPU using CudaMemcpy()

Figure 5: Flowchart for implementing Convolution on CUDA

5. RESULTS AND ANALYSIS

A Simulation setup with NVIDA's GeForce GT 610 Graphics Card was used for implementing DFT and Convolution. The GPU, whose specifications are shown in Table 1, is based on the Fermi architecture with 48 CUDA cores and clock frequency of 1.62 GHz. The CUDA was interfaced with Intel dual core CPU, whose specifications are shown in Table 2, having a clock frequency of 2.60 GHz.

Table 1: GPU Device configuration and specifications

Features	Specifications
Name	GeForce GT610
Compute capability	2.1
Global Memory	2 GB
Multi-processors	1
Number of CUDA cores	48
Total Constant Memory	64 KB
Threads per Block	1024
Shared memory per Block	48 KB
Registers per Block	32768
Blocks per Block	8
Wraps per Multiprocessors	48
Block Dimensions	1024*1024*64
Grid Dimension	65535*65535*65535
Threads per Wrap	32
Clock rate	1.62 GHz

Table 2: Host Machine Specifications

Features	Specifications
Processor	Intel Pentium Dual-Core
Operating System	Windows 10
Clock Speed	2.6 GHz
Memory	2 GB RAM

For parallel computation of DFT/ Convolution on GPU, the recursive programming is performed on the group or block of threads in kernel execution. Convolution is implemented by launching the kernel by specifying the number of threads and thread blocks for real and imaginary values of DFT and then launching the real and imaginary kernels of convolution. The number of blocks and threads per block depends on size of DFT. The output of the kernel is then copied to the CPU using function cudaMemcpy(). Then output of the DFT is printed on the console in CPU.

The GPU is configured to the host Intel Pentium Dual-Core System. CUDA toolkit v6.5 which mainly consists of compiler (nvcc), the required libraries, the visual profiler and the debugger is used for parallel computing. The execution time of the convolution algorithm on both CPU and GPU is calculated and compared. The comparison result, shown in Table 3 demonstrates the increase in speed improvement with increased N.

Table 3: Execution speed comparison of Convolution

N	CPU(us)	GPU(us)	Speed Improvement Factor
4	22055.3	301.3	73
8	27569.2	316.06	87
16	60258.4	317.86	190

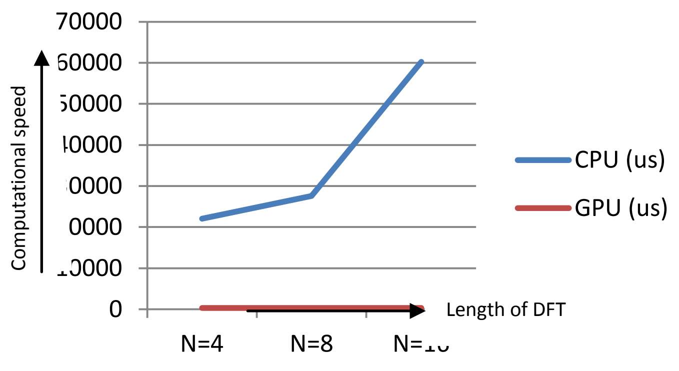


Figure 6: Speed comparison of convolution

6. CONCLUSION AND FUTURE SCOPE

This paper is aimed to implement Convolution using N-point DFT on CPU and CUDA based GPU. The results obtained are compared with respective to execution time. The vast parallelism in GPU has improved the computational speed by many folds by reducing the execution time. The CUDA architecture allows massive parallelism of computations for faster processing on a large data set and recursive operations. The present work implements DFT and Convolution operations which are being widely used for many signal processing applications.

The current work is implemented on 48-core GPU (GeForce GT610), the speed can further be improved by using GPUs having more number of cores. The CUDA architecture thus finds many applications on real-time signal, image, video processing, artificial intelligence and machine learning applications which involves computations on a large set of data and more number of repetitive operations.Our thanks to the experts who have contributed towards development of the template.

7. REFERENCES

[1] Bahri Hayt hem, Hallek Mohamed, Chouchene Marwa, Sayadi Fatma, Atri Mohamed," Fast Generalized Fourier Descriptor for object recognition of image using CUDA" 978-1-4799-2806-4/14 ©2014 IEEE

[2] Meirui Ren, Meihui Ren, Wei ping Zhang1t, Tao Wang, Ning Tian, linbao Li, Longjiang Guo,"An Implementation of the QR Iterations for Finding Eigenvalues of Matrices with CUDA on GPU",2013 International Conference on Mechatronic Sciences, Electric Engineering and Computer (MEC) Dec 20-22, 2013, Shenyang, China

[3] Xiaoxia Qi, Xiao Ma, Dou Li, Yuping Zhao. "Implementation of Accelerated BCH Decoders on GPU", 978-1-4799-0308-5/13/$31.00 © 2013 IEEE.

[4] Mirgita Frasheri, Betim "The use of GPUs in image processing",2nd Mediterranean Coriference on Embedded Computing MECO – 2013.

[5] Yue Zhao and Francis C.M. Lau," Implementation of Decoders for LDPC Block Codes and LDPC Convolutional Codes Based on GPUs", IEEE transactions on parallel and distributed systems.

[6] Alexandros Papakonstantinou1,Karthik Gururaj , John A. Stratton1, Deming Chen1, Jason Cong ,Wen-Mei W. Hwu,"FCUDA: Enabling Efficient Compilation of CUDA Kernels onto FPGAs",2009 IEEE 7th Symposium on Application Specific Processors (SASP).

[7] Mohammad Nazmul Haque,Mohammad Shorif ,"Accelerating Fast Fourier Transformation for Image Processing using Graphics Processing Unit",Journal of Emerging Trends in Computing and Information Sciences Volume 2 No.8, AUGUST 2011.

[8] Jing Wu, Joseph JaJa, Elias Balaras ,"An Optimized FFT-Based Direct Poisson Solver on CUDA GPUs". IEEE transactions on parallel and distributed systems, vol. 25, no. 3, march 2014.

[9] Ke Yan, Junming shan, Eryan Yang,"cuda-based acceleration of the jpeg decoder", 2013 Ninth International Conference on Natural Computation - (ICNC).