International Journal on Recent and Innovation Trends in Computing and Communication
Volume:3 Issue: 5
ISSN: 2321-8169
2741 - 2744

Developing Library of Internet Protocol suite on CUDA Platform

Taralkumar Mistry
Embedded System Design
GTU PG SCHOOL, Gujarat Technological University
Ahmedabad, India
Taral.mistry@gmail.com

Rahul Bhivare
Technical Consultant
CDAC-ACTS
Pune, India
Rahul.bhivare@gmail.com

Abstract **-** Presently, Computational power of Graphics processing Units (GPUs) has turned them into an attractive platform for general purpose application at significant speed using CUDA. CUDA is a parallel computing platform and programming model which has the ability to deliver high performance in parallel applications. In networking world protocol parsing is very complex due to bit wise operation. We need to parse packet at each stage of network to support packet classification and protocol implementation. Conventional CPU with fewer cores is not sufficient to do such packet parsing. For this purpose, we are choosing to build a networking library for protocol parsing, by this way we can offload compute intensive protocol parsing task on CUDA enable GPUs which can optimize usage of CPU and improve system performance.

Keywords **-** *CUDA, GPGPU, Packet Parsing.*

*****

I. INTRODUCTION

A major challenge in todays embedded world is high performance computing and real time performance. This is difficult to achieve with the even more powerful CPU. Also, modern GPUs are on the leading edge of increasing chip level parallelism by supporting hundreds of cores on a single chip. This degree of hardware parallelism reflect the fact that GPU architecture involves to feet need of real time computational application. The main objective of parallel processing is high performance by reducing execution time, improve efficiency and better utilization of resources. To attain such parallel processing, we describe our system built on CUDA (Compute Unified Device Architecture) platform

As the internet evolves, there is growing need for nontrivial packet parsing at all stages of network infrastructure [6]. Packet parsing is important in order to identify what is flow of packet and to implement quality of services goal. The packet parsing task is difficult due to bit parsing. In bit parsing, the operation is performed at bit level. Using INTEL CPU with 32 bit and 64 bit, operate at bit level would be a waste of power and cycle which is not necessary. For that we are going to offload such protocol parsing to CUDA enable GPU.

A. GPU Accelerated Computing

GPU accelerated computing is use of GPU together with CPU to accelerate applications. GPU accelerated computing offer great performance by offloading compute intensive part of the application to GPU [3]. An application which demands more parallelism will be offloaded to GPU where as part of the application which demands lower parallelism still runs on the CPU. Ultimately, application run faster.

The main difference between CPU and GPU is to compare how they process tasks. A CPU consists of a few cores optimized for sequential serial processing, while a GPU has massive parallel architecture consist of thousands of cores called processing element for handling multiple task simultaneously. Figure 1 shows, difference between CPU and GPU cores.

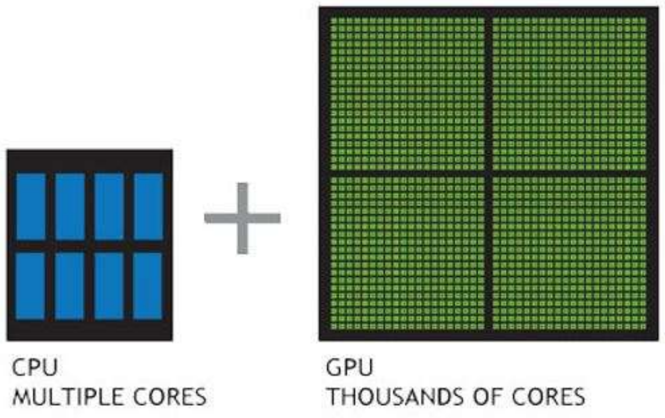

Figure 1 Difference between CPU and GPU cores [3]

B. GPGPU

GPGPU is a general purpose computing, on the GPU. Till the time GPU is used for graphics intensive application, but with the advent of technology by placing many cores on a single chip, we can use GPU for any complex task. We can use GPU for any general purpose application which need high parallelism. GPGPU is a term used in high performance computing (HPC) to accelerate application [5]. In GPGPU programming technique, programmers can use GPU pixel shavers as general purpose single precision FPUs. Off-chip plays an important role in GPGPU as all threads interact with each other using off chip memory. GPGPU implementation exhibits two properties. First data parallelism where processor can execute operation on different data elements simultaneously and the second is throughput mean it can process so many data elements and exhibits parallelism.

This paper presents an overview of CPU-GPU heterogeneous computing, GPGPU and approach for offloading protocol parsing on CUDA enable GPU. This paper is organized as follows. Section II briefly introduces CUDA and protocol parsing. Section III further explains proposed system. Section IV shows the result of Implementation. Finally, Section V concludes the paper.

II. OVERVIEW OF CUDA AND PACKET PARSING

The chapter introduces two subsection. First about CUDA, its programming model and memory model. Second about overview of protocol paring.

A. CUDA

NVIDIA's Compute Unified Device Architecture (CUDA) is a software platform for massively parallel high performance computing on NVIDIA's powerful GPUs. NVIDIA CUDA technology is fundamentally new computing architecture that enables the GPU to solve complex computational problem [2]. It gives computationally intensive applications access to the processing power of NVIDIA GPU through the new programming interface. Software development is strongly simplified by standard C language with some extension. NVIDIAs software CUDA programming model effectively uses GPUs which could be harnessed for the task other than graphics achieving Gigaflops of computing power.

1) CUDA programming model

NVIDIA simplifies a programming model in which burden of managing threads is removed. This is an important feature of CUDA in which application programmers don't write explicit threaded code. Hardware thread manager handles threading automatically. Automatic thread management is vital when multi-threading scales to thousands of threads. NVIDIAs card can manage as many as concurrent threads and these are lightweight threads in the sense that each thread can operate on a small piece of data.

CUDA is a parallel programming model and at its core are three key abstractions- a hierarchy of thread groups, shared memories and barrier synchronization [4]. This abstraction simply provides fine-grained data parallelism and thread parallelism, nested with coarse-grained data parallelism and task parallelism. They guide programmer to partition problem into a coarse sub problem that can be solved independently in parallel and then into finer piece that can be solved cooperatively in parallel. The CUDA programming model automatically manages the threads and it significantly differs from single threaded CPU card and some extent even parallel code. Figure 2 shows a programming model of CUDA.

CUDA assume that CUDA threads may execute on a physically separate device that operate as a co-processor to the host running C program. This is the case when kernel executes on a GPU and the rest of C program execute on the CPU. Also, both host and device maintain their own DRAM, refer to as host memory and device memory.

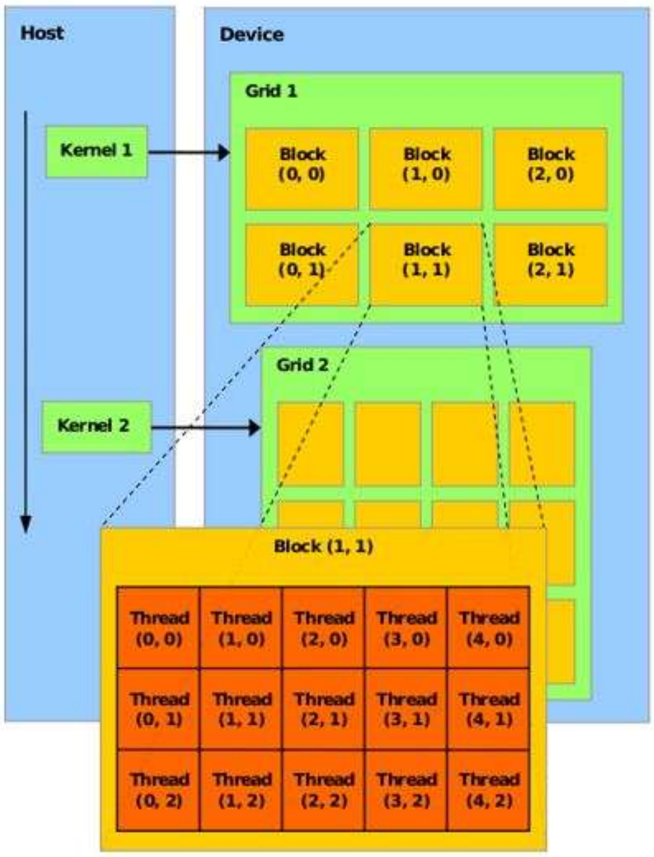

Figure 2 CUDA Programming Model [5]

2) CUDA Memory Layout

CUDA thread may access data from multiple memory space during their execution. Each thread has private local memory. Each thread block has shared memory visible to all threads of the block and with the same lifetime as a block. All threads have access to global memory. Also, two additional read only memory spaces accessible to all threads: constant and texture memory space. Global, texture and constant memory are optimized for different memory usages. Texture memory also offers different addressing modes and data filtering for some specific data format. Global, constant and texture memory are persistent across kernel launched by the same application.

CUDA consists of basically five types of memory these are texture, constant, global, local and shared memory. Figure 3 shows CUDA memory layout which consists different types of memory. Global and shared memories are introduced in CUDA, these two are most important and commonly in use. Another three are used to improve performance. Local memory is on chip memory, which only allows threads within block to access the data. Constant memory has the feature of cache memory, its accessing speed is fast. Global memory is the main memory of the GPU, any data communication between CPU and GPU is done through global memory. Also outcome of any block will be stored in global memory. Off chip memory plays an important role in performance of the system.

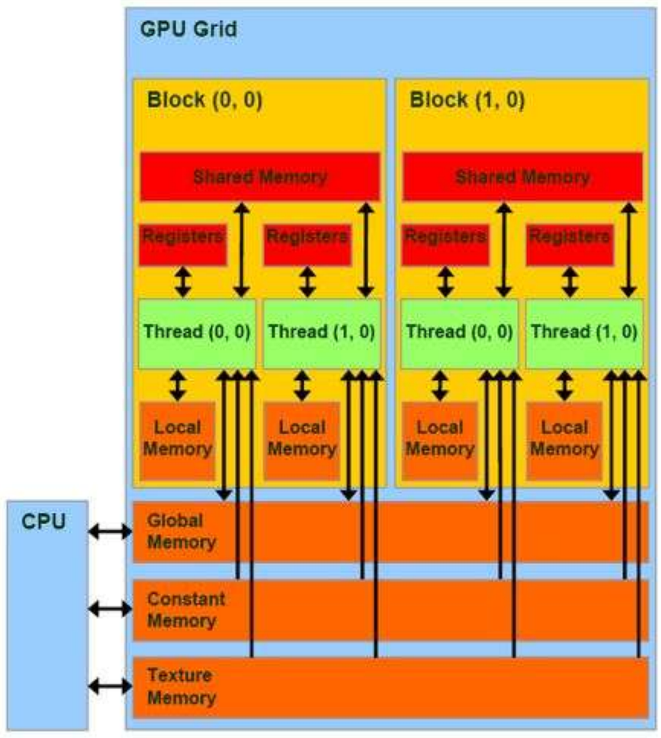

Figure 3 CUDA Memory Layout

B. PACKET PARSING

All network devices must parse packet to decide how a packet should be processed. Packet parsing is necessary at all points in networking, to support packet classification and for security. To implement protocol we need to parse packet. Packet parsing has an important role in end to end communications [6].

Parsing is the process of identifying headers and extracting fields for processing by subsequent stage of device [6]. Each packet consists of stacking of header, data payload and optionally stack of trailers. Protocol parsing is sequential: each header is identified by preceding header, requiring header to be identified in the sequence.

Current high speed network demands more and more data from the application layer. Also, in networking protocol parsing is difficult due to bit parsing. The bit mask is used to extract information embedded at bit level, which is useful to extract field of header of protocol in a packet. For example, TCP header contains address of destination, source address with a number of flags embedded in its header field, we can consider them to store within data width of short. Also, modern computer encode data in little endian whereas all network packets are encoded in big endian. So parsing should be done effectively and fast.

III. PROPOSED SYSTEM

The whole system is based on client-server model. In system, one PC will send a data packet to another INTEL CPU via Ethernet cable. INTEL CPU gives this information to CUDA enable GPU. The GPU will receive this packet and parse it. By this way compute intensive task of CPU can be offloaded to CUDA enable GPU.

We are proposing an idea to build libraries for protocol parsing suite on CUDA platform. By using parallel computing capabilities of CUDA, we can improve system performance. CUDA enables GPU is working as a streaming processor and we can offload protocol parsing task to the GPU.

The whole work is divided into three phases:

Phase 1: where parsing is done by INTEL CPU

Phase 2: where parsing is done by CUDA GPU

Phase 3: Possible test cases

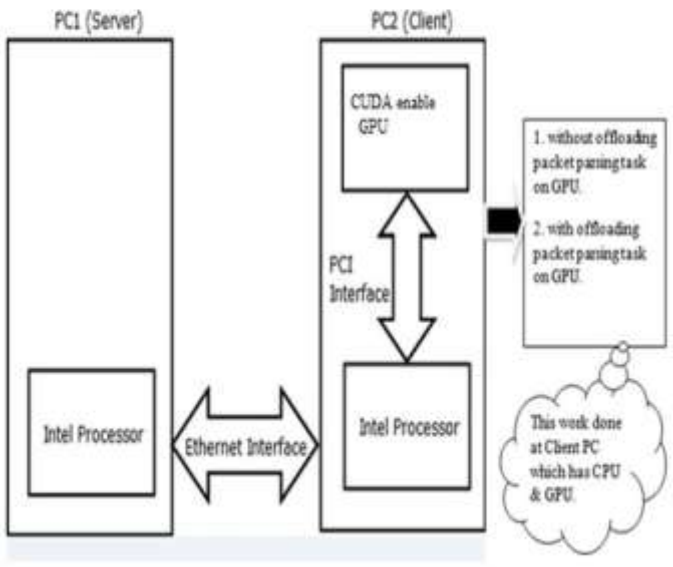
PC1 (Server)
PC2 (Client)
CUDA enable GPU
PCI Interface
Intel Processor
Ethernet Interface
Intel Processor
1. without offloading packet parsing task on GPU.
2. with offloading packet parsing task on GPU.
This work done at Client PC which has CPU & GPU.

Figure 4 Proposed System

Phase 1: In this phase, the packet is sent from sender PC to another PC through tool like Iperf. At receiver socket program is able to catch those packets and try to parse that packet, and it will identify types of packet. At the same time we analyzed system performance mean CPU usage. Also will check packetized data at the receiver. For example, the sender sends an image, then the receiver should receive it correctly and display that image. Here packet parsing task is done by INTEL CPU.

Phase 2: In this phase, the actual implementation has on CUDA enabled GPU. Here the idea is to parse packet using GPU instead of the CPU. All procedure is as above in phase 1. GPU is working as a streaming processor, mean programmer has to decide that how many cars he wants to use for a particular task among all available core. At a same time we analyzed system performance. By this way we can effectively parse packet, and system performance would be increased.

Phase 3: once all functionality has been achieved using GPU, we will have to test thing using different cases:

Case 1: Receive data and verify types of data.

Case 2: Packet should send random and receive correctly

Case 3: Packet lost checks

At the receiver side, we will have socket code which will test all possible test cases discussed above. After verifying all cases we can say that idea is working properly.

A. FLOWCHART OF SYSTEM

First, we need to invoke kernel mean CPU will call GPU to execute tasks. Once the kernel is invoked, then we will have to allocate memory in GPU uses cudaMalloc () API. As each GPU has its own dedicated memory, the programmer has to decide and allocate required memory. After that, data has been copied out from the CPU to GPU using cudaMemcpyHostToDevice () API. Once the packet has transferred to CUDA enable GPU than each packet will process by one thread as followed by SIMT (Single Instruction Multi-Threading) and decide which type of packet is. After deciding the type of packet, particular packet will process and the result will send back to the host. Figure 5 shows the flow of the system and how each packet has been proceeded by the GPU.

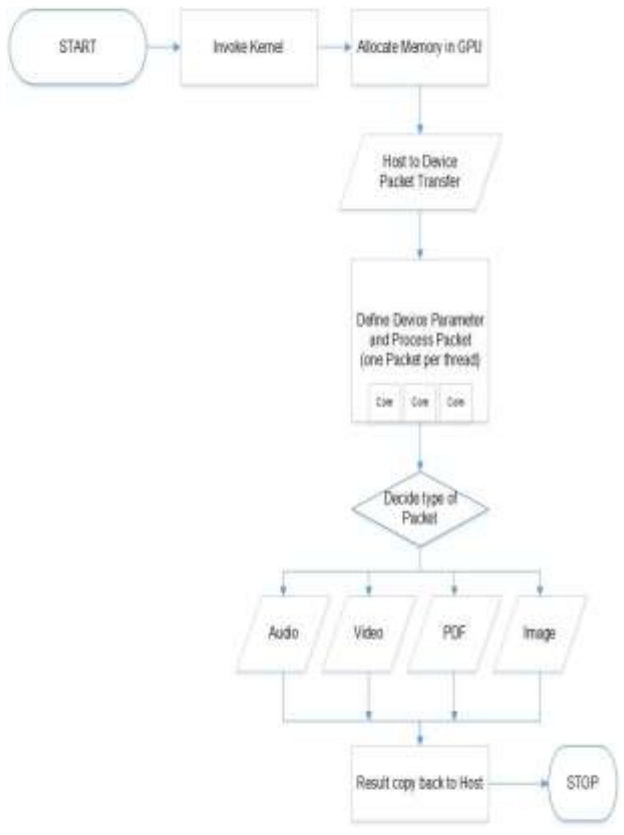

Figure 5 Flowchart of System

IV. RESULTS

In this work, Different type of data like audio, video, image and pdf is processed by NVIDIA ZOTAC 8400GS which has 16 CUDA cores. I used Microsoft visual studio 2010 as a programming platform. The first video file is sent to the receiver and then I analyzed packet processing on both CUDA based GPU and CPU.

Table 1 lists the results of comparisons of CUDA technology and CPU implementations. Figure 6 shows the Improvement in system performance in terms of usage. From Table 1 and Figure 6, I can say that CUDA based packet processing is efficient and fast.

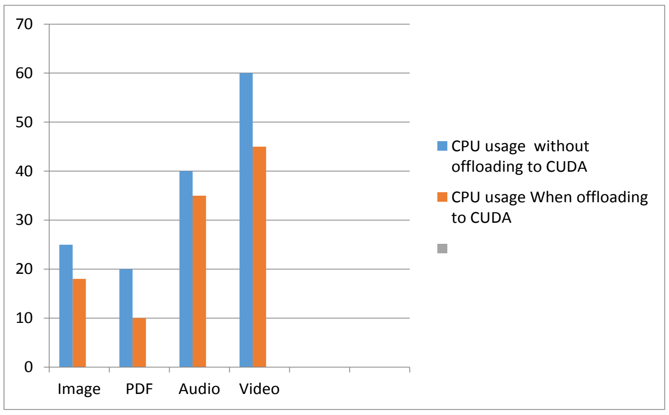

Figure 6 Comparison of CPU Usage

Table 1 Performance Analysis

	CPU usage without offloading to CUDA	CPU usage when offloading to CUDA
Image	25	18
PDF	20	10
Audio	40	35
Video	60	45

V. CONCLUSION

From this paper we conclude that CUDA enable GPU is able to execute any complex task very effectively. After building libraries, we can offload protocol parsing task to CUDA enable GPU. System performance would be increased. So we can let CPU free to do any other urgent work.

VI. REFERENCES

[1] CUDA Home Page, http://deveper.nvidia.com/object/cuda.html.

[2] GPGPU, "General-Purpose Computation Using Graphics Hardware," http://www.gpgpu.org.

[3] "Gpu computing", http://www.nvidia.com/object/what-is-gpu-computing.html

[4] Er. Paramjeetkaur and Er. Nishi, "A Survey on CUDA", International Journal of Computer Science and Information Technologies, Vol. 5 (2) , 2014, 2210-2214.

[5] Glen Gibb, George Vargashe, Mark Horowitz, "Design Principal for Packet Parser", IEEE Symposium,21-22 Oct. 2013, Pages 13-24.

[6] Manuel Ujaldon, Nvidia CUDA fellow, "High Performance Computing and Simulation on the GPU using CUDA", IEEE, 2-6 July 2012, Page1-7.

[7] Ching-Lung Su, Po-Yu Chen, Chun-ChiehLan, Long-Sheng Huang and Kuo-Hsuan Wu,\emph{"overview and comparison of OpenCL and CUDA technologies for GPGPU"}, IEEE Asia Pacific Conference, 2-6 Dec. 2012 , Page 448-451.

[8] John Nickolls, Michael garland, NVidia, Scott Morton, "Parallel Computing Experience with CUDA", IEEE, July-Aug. 2008, Page 13-27.

[9] Wen-Mei Hwu, Christopher Rodrigues, Shane Ryoo and JohnStartton,\emph{ "Compute Unified Device Architecture Application Suitability"}, IEEE, May-June 2009, Page 16-26.

[10]