ISSN: 2321-8169
2650 - 2653

Developing Library for Transport Layer of Internet Protocol Suite on CUDA platform

Chintan Prajapati
Embedded System Design
GTU PG SCHOOL, Gujarat Technological University
Ahmedabad, India
rock.chintan@gmail.com

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

Abstract **-** Presently, the computational power of graphics processing units (GPUs) has turned them into attractive platforms for general-purpose applications at significant speed using CUDA. Compute Unified Device Architecture (CUDA) programmed, Graphic Processing Units (GPU) is rapidly becoming a major choice in high performance computing (HPC). Hence, the number of applications ported to the CUDA platform is growing high. So, a major challenge in today's embedded world is high performance computing and to attain high precision and real time performance-which is difficult to achieve even with the most powerful CPU. In the networking world, Packet parsing is a complex task due to bit wise operation. So we can offload packet parsing task on the CUDA enable GPU. For this purpose, we are choosing to build networking library prototype, to boost the processing speed of networks on CUDA compatible GPUs. In response, we propose to develop the libraries for parsing transport layer of internet protocols on NVIDIA CUDA parallel processing platform (NVIDIA CUDA enabled GPU). With this, we can offload the protocol parsing task of Intel CPU, optimize the CPU usage and increase the performance efficiencies.

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

*****

I. Introduction

Graphics Processing Units (GPUs) and Central Processing Units (CPUs) have often been compared to one another, and not without reason. Both have processors often with multiple cores, caches, internal buses, registers, ALUs, and more [1]. CPUs are general purpose machines, while GPUs are specifically dedicated to being able to process many similar operations in parallel, allowing them to render a screen very quickly in comparison to a CPU [1]. Till the time GPUs are used for the graphics intensive application, but with the advance of technology by placing many cores on a single chip, we can use GPU for any complex task mean we can do general purpose computing on GPU which is called GPGPU.

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. 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.

The protocol parsing task is difficult because of bit parsing and current high speed network demands. For that we need to parse packet very fast and efficiently to satisfy such demand. CPU with fewer cores is not sufficient to do such things. So for that we are going to offload such protocol parsing to CUDA enable GPU.

A. GPGPU

GPGPU stands for "General Purpose computing on Graphics Processing Units." General-purpose computing on graphics processing units (GPGPU) is the utilization of a graphics processing unit (GPU), which typically handles computation only for computer graphics, to perform computation in applications traditionally handled by the central processing unit (CPU) [2]. GPGPU implementation has two properties. First data parallelisms 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.

II. Overview of PACKET PARSING and CUDA

A. 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 [3]. For example, a router examines the IP destination address to decide where to send the packet next, and a firewall compares several fields against an access-control list to decide whether to drop a packet [3].

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

Packet parsing is a key bottleneck in high speed networks because of the complexity of packet headers. Packets often contain many more headers. These extra headers carry information about higher level protocols (e.g., HTTP headers) or additional information that existing headers do not provide [3]. Each incoming packet must go through some sort of parsing to examine and understand what it is as well as its requirements, and then it must be classified, or handled according to its type and its required processing. Parsing therefore is the first analysis and action done on the packet content. Parsing can be very simple, trivial, and unnoticed during packet processing, or it can be a real and complex task that sometimes requires a unique language to describe the process. The task of parsing is sometimes even carried by a unique, dedicated processing element [4].

Parsing is basically identifying the relevant fields in the incoming packets, according to their place and type, and picking the field's values for continuing the parsing process, or using these values for classification. A simple parsing example, in an IPv4 packet would be to detect its destination IP address, which is easy, since it is a fixed length field in the IP header, always at the same offset of the packet [4].

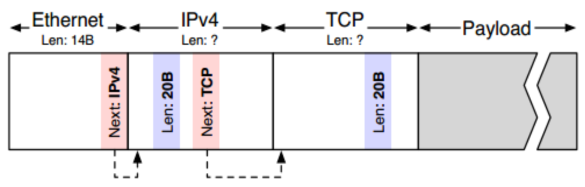


Figure 1 Parsing TCP packet [4]

As shown in figure 1. The large rectangle represents packet. Processing begins at the head of Packet that is Ethernet header. From this header parser extract the header length and next header type. It will read the next header type that will be IPv4. After that it is going to parse IPv4 header from previous header. It will extract all information about that header like IP address of source and destination, next header type, header length, data payload length. After it is going to parse next header, which is TCP header. From this header, it will extract all information like source and destination port, length of the header, sequence number, acknowledgment number etc. This process repeats until all headers are processed.

B. *CUDA*

Compute Unified Device Architecture (CUDA) is a new hardware and software architecture created by NVIDIA for designing and dealing with parallel computations on the GPU. The initial CUDA SDK was made public on 15 February 2007, for support was later added in version 2.0. CUDA works with all NVIDIA GPUs from the G8x series onwards, including Geforces, Quadro and the Tesla line. The release of GPU programming platform CUDA offers a highly parallel computation and flexible programmable platform [5]. CUDA application programming interface (API) enables software developers to access the GPU and also enables researchers to design programs for both CPU and GPU with a C like programming language, Without basic knowledge of computer graphics. 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 [5].

1) CUDA programming model

This is an important feature of CUDA in which application programmers don't write explicit threaded code. Hardware thread manager handles threading automatically. NVIDIA's 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.

A kernel is executed by a grid (decomposition of a problem into sequential steps), which further contain blocks (decomposition of grids into parallel blocks called (CTAs), these blocks again contain threads (decomposition of blocks into parallel elements). A thread block is a collection of threads that can share data through shared memory and synchronized to their execution. But threads from different blocks operate independently. The CUDA programming model automatically manages the threads and it significantly differs from single threaded CPU code and some extent even parallel code. Figure 2 shows a programming model of CUDA [5].

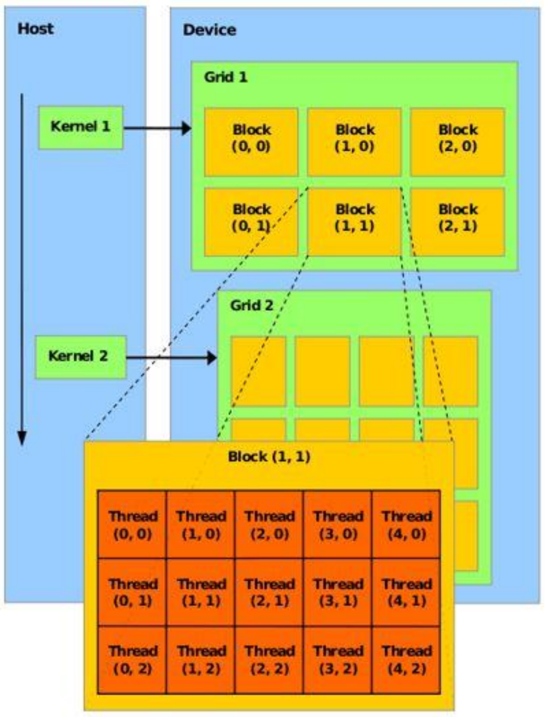


Figure 2 CUDA Programming Model [5]

2) *CUDA Memory Layout*

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 [5].

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.

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.

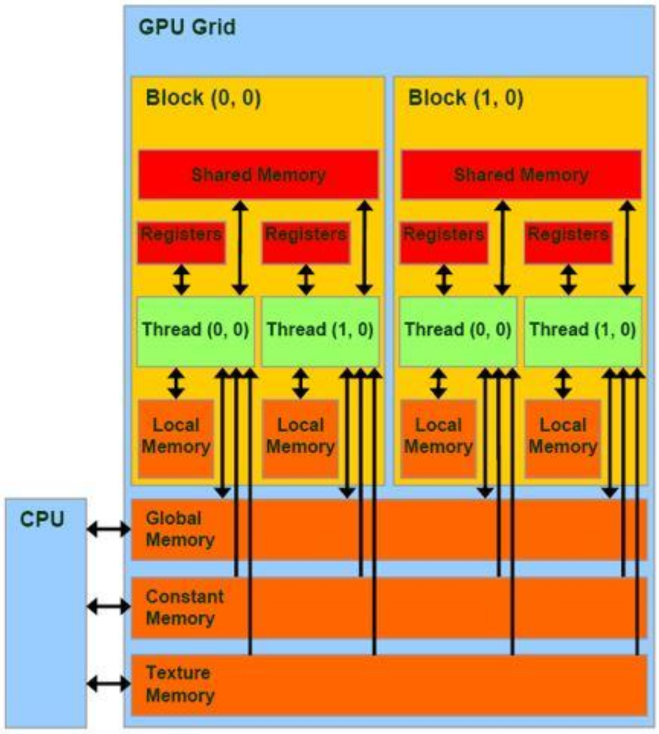


Figure 3 CUDA Memory Layout [5]

III. PROPOSED SYSTEM

The whole system is based on client-server model. In system, there are two PC one is sender and another one is receiver. Sender PC has only Intel CPU, it will transmit multiple data (packet) to receiver PC. Now on the receiving PC side, it has GPU card along with Intel CPUs. Receiver PC captures those data (packets) and further processes using Intel CPU. Receiver's CPU usages will increase. So optimize usages of Intel CPU we used NVIDIA GPU (CUDA enabled GPU). So with this, we can offload the protocol parsing task of Intel CPU. Again, offloading the parsing task on the GPU can optimize the computational performance of the system. The packet will transmit from one PC to another PC using Network Internet protocols.

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.

Flow of proposed system is shown in figure 4.

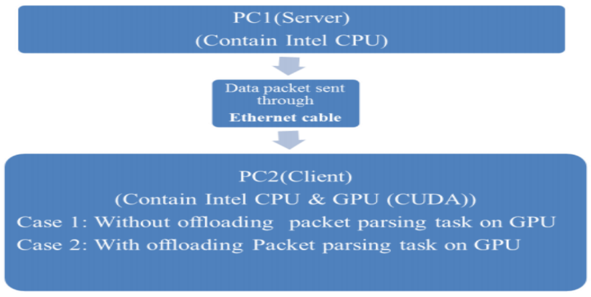
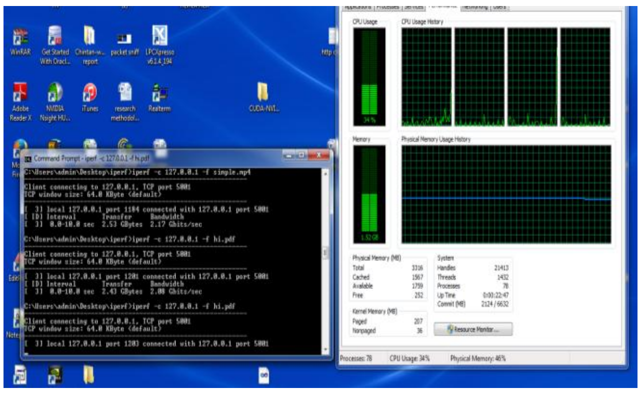

PC1(Server)
(Contain Intel CPU)
Data packet sent through Ethernet cable
PC2(Client)
(Contain Intel CPU & GPU (CUDA))
Case 1: Without offloading packet parsing task on GPU
Case 2: With offloading Packet parsing task on GPU

Figure 4 Flow of Proposed System

Phase 1: In first phase, two PCs connect with Ethernet cable and some packets are sent from one PC to another PC using some tools like iperf. At the receiver side one code is available on windows platform through which we can catch those packets, count those packets and identify their types that particular which type of packet it is. Means it can be TCP, UDP, HTTP, Ethernet, ARP, and of any type. This identification and counting of packets are done on the receiver side through socket programming code in visual studio on windows platform by using Winsock. By the same time I also analyzed some parameters like CPU usage and all that. I checked that how these different types of packets affected to CPU usage. In this case i sent different type of packet means I sent pdf file and that time I checked the system usages, it's around 32% to 34%.



```
C:\Users\admin\Desktop\iperf>iperf -c 127.0.0.1 -f simple.mp4
------------------------------------------------------------
Client connecting to 127.0.0.1, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 1184 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  2.53 GBytes  2.17 Gbits/sec

C:\Users\admin\Desktop\iperf>iperf -c 127.0.0.1 -f hi.pdf
------------------------------------------------------------
Client connecting to 127.0.0.1, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 1201 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  2.43 GBytes  2.08 Gbits/sec

C:\Users\admin\Desktop\iperf>iperf -c 127.0.0.1 -f hi.pdf
------------------------------------------------------------
Client connecting to 127.0.0.1, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 1203 connected with 127.0.0.1 port 5001
```

Figure 5 result of Phase-1

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 cores 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 usages would be decreased.

Whatever packets are coming on CPU, that should be passed to GPU memory. Now from CPU, GPU kernel is invoked and process all the packets. And parse all headers of packets and give the result back to the CPU.

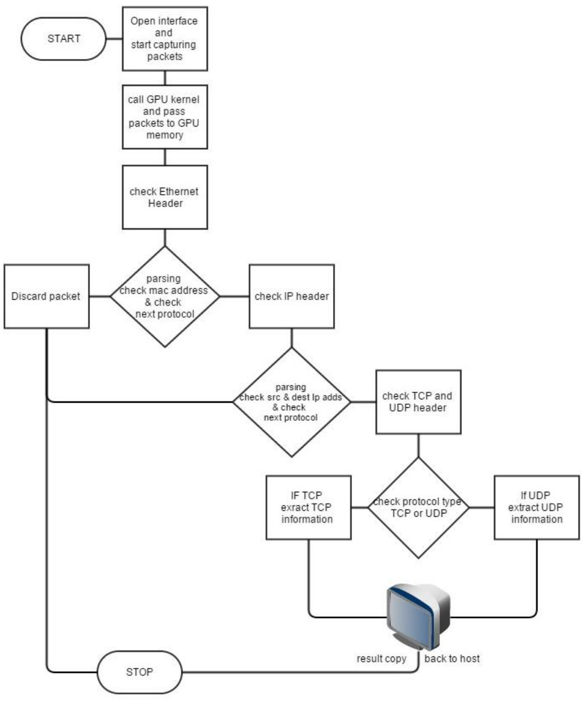
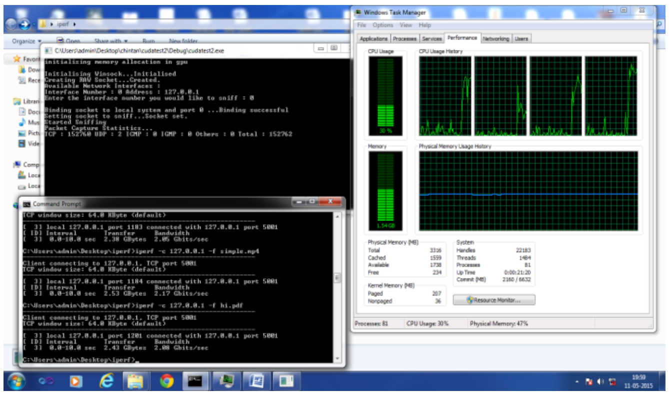

START
Open interface and start capturing packets
call GPU kernel and pass packets to GPU memory
check Ethernet Header
parsing check mac address & check next protocol
Discard packet
check IP header
parsing check src & dest Ip adds & check next protocol
check TCP and UDP header
check protocol type TCP or UDP
IF TCP exract TCP information
If UDP extract UDP information
result copy
back to host
STOP

Figure 6 Flow chart

For Phase-2, I sent same file and check system usages and its about 30% to 32%.



```
C:\Users\admin\Desktop\chintan\cudatest2\Debug\cudatest2.exe
initializing memory allocation in gpu

Initialising Winsock...Initialised
Creating RAW Socket...Created.
Available Network Interfaces :
Interface Number : 0 Address : 127.0.0.1
Enter the interface number you would like to sniff : 0

Binding socket to local system and port 0 ...Binding successful
Setting socket to sniff...Socket set.
Started Sniffing
Packet Capture Statistics...
TCP : 152760 UDP : 2 ICMP : 0 IGMP : 0 Others : 0 Total : 152762
```

```
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 1183 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  2.38 GBytes  2.05 Gbits/sec

C:\Users\admin\Desktop\iperf>iperf -c 127.0.0.1 -f simple.mp4
------------------------------------------------------------
Client connecting to 127.0.0.1, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 1184 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  2.53 GBytes  2.17 Gbits/sec

C:\Users\admin\Desktop\iperf>iperf -c 127.0.0.1 -f hi.pdf
------------------------------------------------------------
Client connecting to 127.0.0.1, TCP port 5001
TCP window size: 64.0 KByte (default)
------------------------------------------------------------
[  3] local 127.0.0.1 port 1201 connected with 127.0.0.1 port 5001
[ ID] Interval       Transfer     Bandwidth
[  3]  0.0-10.0 sec  2.43 GBytes  2.08 Gbits/sec

C:\Users\admin\Desktop\iperf>_
```

```
30 %
1.54 GB

Physical Memory (MB)          System
Total        3316             Handles        22183
Cached       1599             Threads        1484
Available    1738             Processes      81
Free         234              Up Time        0:00:21:20
                              Commit (MB)    2160 / 6632
Kernel Memory (MB)
Paged        207
Nonpaged     36

Processes: 81    CPU Usage: 30%    Physical Memory: 47%

19:59
11-05-2015
```

Figure 7 result of Phase-2

IV. Conclusion

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

V. References

[1] http://cmsc411.com/gpgpu/overview

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

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

[4] Book- Ran Giladi, "Network Processors - Architecture, Programming, and Implementation"

[5] "A Survey on CUDA"- Er. Paramjeet kaur and Er. Nishi, Department of Computer Science and Engineering, DAV University, Jalandhar, Punjab, India - (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 5 (2) , 2014, 2210-2214 IEEE.

[6] "Survey on Transport Layer Protocols: TCP & UDP"- Santosh Kumar, Sonam Rai, Graphic Era University, Dehradun (India) - International Journal of Computer Applications (0975 – 8887) Volume 46– No.7, May 2012 IEEE.

[7] "The Architecture and Evolution of CPU-GPU Systems for General Purpose Computing"- Manish Arora, Department of Computer Science and Engineering University of California, San Diego La Jolla, CA 92092-0404.

[8] "EFFICIENT PARALLEL PROCESSING BY IMPROVED CPU-GPU INTERACTION"- Harsh Khatter, Department of Computer Science and Engineering, ABES Engineering College, Ghaziabad, India and Vaishali Aggarwal, Department of Computer Science and Engineering, KIET, Ghaziabad, India.

[9] "Towards using the Graphics Processing Unit (GPU) for Embedded Systems"- Daniel Hallmans, ABB AB Ludvika, Sweden and Mikael ˚Asberg, Thomas Nolte, MRTC/M¨alardalen University P.O. Box 883, SE-721 23 V¨aster˚as, Sweden.

[10] "SCALABLE PARALLEL PROGRAMMINGFOR HIGH-PERFORMANCE SCIENTIFIC COMPUTING"- David Luebke, NVIDIA Corporation.

[11] Book-"CUDA C PROGRAMMING GUIDE" - Design Guide by Nvidia.