Parallelizing CPU-GPU Network Processing Flows

Anup Nair
Dept. of Computer Engineering and IT
College of Engineering, Pune
Pune, India
nairas18.comp@coep.ac.in

Amit Joshi
Dept. of Computer Engineering and IT
College of Engineering, Pune
Pune, India
adj.comp@coep.ac.in

2022 International Conference on Innovative Trends in Information Technology (ICITIIT) | 978-1-6654-0670-3/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICITIIT54346.2022.9744209

***Abstract*—Network processing has traditionally been a CPU-intensive operation where every device in the network has to do packet processing. With the upcoming needs for a digital world and rising technologies like 5G, the demand for faster processing has dramatically increased. In such cases, using only the CPU for network processing across core devices and edge devices can become a major bottleneck. This work aims to explore the use of GPUs for network processing and exploiting data-level parallelism in network-processing operations to speed up the overall network. The work throws light on how data transfer overheads can be minimized using CUDA Streams and achieves a 2x performance improvement with respect to synchronous data transfer. The subsequent part of this work deals with the implementation of packet switching on GPUs with the help of Bloom Filters. The exponentially increasing execution time on the CPU with respect to the number of packets is reduced to a constant execution time on GPU.**

***Index Terms*—GPU, Network Processing, CUDA Streams, Bloom Filters**

I. Introduction

With the advent of the COVID-19 pandemic, the whole world transitioned into a digitally connected phase. Offices, colleges, and almost every area of life were slowly brought to the internet. Statistics show a sharp rise in the consumption of Internet-based applications like video-conferencing, social networking, online orders, and many more. With so many important jobs relying on the internet, it becomes necessary to optimize it to meet the ever-increasing demands. According to Cisco's annual internet report, it is expected that the fixed global internet broadband speed is about to reach 110 Mbps which is about 2.5 times more than in 2018 [1]. Add to this new technology like 5G, which claims to increase throughput by a factor of 1000 times over 4G with an estimated 575 Mbps throughput, the need for better network processing has become an emerging challenge. It is also estimated that about 70% of the world's population will be on mobile connectivity by the end of 2023.

Graphics Processing Units (GPUs) have evolved from being used just as multimedia processing devices to general-purpose computing devices (GPGPUs) [2]. GPGPUs have since been used in various domains like machine learning and statistical computing [3]. Major mobile platforms like smartphones and laptops introduce better GPU hardware every year, most of which are used for dedicated media applications only. Major focus can be shifted to using these devices for network computing at the edge. With open APIs in the form of Compute Unified Device Architecture (CUDA) and Open Computing Language (OpenCL) available, it becomes easier to offload complex computation procedures directly to GPUs [4]. Besides, wireless networks have evolved substantially in the past decade, and research on using GPUs in such networks has been explored recently [5]–[7].

This work aims to explore various network processing operations and study their inherent structure to exploit data-level parallelism. This helps to offload these operations from CPU to GPU and has a promising potential due to the way GPUs can perform parallel computing and can help achieve much better performance from networking devices [8]. The two key areas of this work are to reduce data transfer overhead from the host to the device with the help of asynchronous APIs and to explore the performance of packet switching on a GPU with the help of space-efficient and fast data structures like Bloom Filters [9], [10].

This work addresses the usage of CUDA Streams to minimize data transfer overheads and evaluate the performance of Bloom Filters for packet switching. The rest of the paper is organized as the literature review in Section II followed by the proposed methodology in Section III. Section IV throws light on the experimental setup, achieved results, and its discussions, followed by the conclusion in Section V.

II. Literature Review

The domain of networking with GPUs has had significant research done on it in the past. Major targeted operations for research were lookup, either on Layer 2 or Layer 3 of the IP model [11]. Han et al. implemented a GPU-based software router called PacketShader that gives a 4x better performance in packet routing than a traditional CPU-only implementation [12]. Deng et al. proposed Hermes, a GPU-based IP routing implementation that achieves about 72.9% improvement over CPU-only routing for delay variance [13]. Shojania et al. implemented a similar tool for network encoding called Nuclei [14]. This work utilized 90% of the GPU limit for encoding and decoding operations. The performance was also compared to CPU-only mode, and the results from the GPU outperformed the CPU-only implementations. Both of the above implementations use CUDA as the underlying tool for

978-1-6654-0670-3/22/$31.00 ©2022 IEEE

offloading procedures to GPUs. Silberstein et al. implemented GPUnet, which provides a socket abstraction and networking APIs for GPU programs [15].

To explore how CUDA works with high-level languages like Python as opposed to a compiled language like C/C++, Holm et al. studied the performance, energy efficiency, and usability of CUDA with Python [16]. Their research shows that as GPUs spend significant time in numerical computing, a language like Python can be used for maintaining program flow without performance degradation.

Bloom Filters are very efficient data structures that are commonly employed in Layer 2 lookups in network devices. Elmori et al. performed a comparison of different variants of Bloom Filters and extended the same study on the MapReduce paradigm [17]. The results from the paper show that using MapReduce for Bloom Filter implementations is limited by data transfers between the map and the reduce phases. Buhler et al. studied the performance of Bloom Filters on GPU engines [18]. The analysis is performed on a genome sequencing dataset from BLAST and uses on-chip shared memory instead of texture memory for storing the Bloom vectors. The work shows that partitioning sets into subsets could improve the false-positive rate for the Bloom Filter operations. Iacob et al. show usage of GPUs for information retrieval using Bloom Filters [19]. This work shows a 20x improvement in querying information over a CPU-based implementation. Since Bloom Filters use multiple hash functions, the performance of hash functions on GPU is also a potential area for speedup. Huang et al. proposed DyCuckoo, which is an implementation of dynamic hash tables on GPUs [20]. The work shows that the find and delete operations can be implemented with an upper limit of two lookups. Hayashikawa et al. proposed Folded Bloom Filters, which provide higher throughput of about 25.6 Gbps [21]. The result is achieved on a pattern set of the size of 744 million with a low false-positive rate. The results show a 12.2% speedup in the case of block ciphers over a typical shared memory implementation and achieve large-scale parallel hashing. Analysis of Folded Bloom Filters was made by Reviriego et al., and Byun et al. [22], [23]. The works have shown that Folded Bloom Filters have low search failure rates over hash tables. Lim et al. analyzed the use of hashing to perform an IP lookup. The work achieves one route lookup in two memory accesses in parallel and generates space-efficient route tables [24]. The results were based on the MAE-WEST router data.

[25]–[27] describes Network coding using GPUs. Shojania et al. achieved a 4x speedup over CPU-only network coding. Xiaowen et al. extended this further and achieved a higher throughput [28]. Robinson et al. studied the energy consumption for this proposed methodology, and the work claims to reduce energy utilization and medium time product by 77.2%. More recent advancements like Optical Label Switching in wireless networks and One-Level Entropy Hashing for packet classification have made use of GPU-based implementation techniques [29], [30].

III. Proposed Methodology

This section explains the optimizations that make GPU offloading a better option over current implementations. It also describes how GPUs can perform network computations. The two explored areas are:

1) Stream based memory transfers from host to device
2) Bloom Filter based packet switching

A. Optimizations on the GPU

The first part of the work focuses on the properties of various network processing operations that can be refactored to run on the GPU. The main properties that are desired for parallel implementation are:

- No interdependence between packets
- Data level parallelism
- Parallel flows within processing pipelines

The flow that is followed on systems where GPUs are used to accelerate computation is as follows:

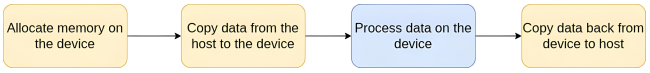

Fig. 1. Processing flow of a GPU routine.

While offloading packet processing to GPUs can result in faster speeds, there is an overhead in copying the packets to the GPU memory. The processing time on GPUs should be larger than the data transfer time for an operation to be beneficial when run on a GPU. This implementation uses CUDA for performing GPU processing. CUDA architectures have several CUDA cores, with each of them having different streaming multiprocessors with shared memory, instruction unit, constant, and texture memory. These are on-chip memory units. All the CUDA cores share an off-chip device memory. The live stream of packets is captured and copied to the device from the host for further processing. Unlike matrices, where the complete data is available in the main memory, which is then copied to the device using a single Application Programming Interface (API) call, a live packet stream needs to be copied in multiple API calls. This operation could result in a more significant portion of time being spent in transferring data across device and host.

The implementation uses trafgen to generate traffic on the ethernet interface [31]. A packet sniffer using the libpcap library captures these packets and copies them to the GPU memory. With the standard CUDA memory copy operation on the default stream, the behavior is synchronous. The portion of the time required to transfer data in such cases is calculated for different packet rates. Since the packets form a stream, using an asynchronous copy operation can result in much better throughputs. CUDA offers Streams and Events to achieve concurrency among kernel routines. A set of multiple streams is created to ensure concurrent transfer of data to the GPU for cases where the packet transfer rate exceeds the copying and processing time on the GPU. Usage of the above techniques has improved the time taken for data transfer. Also, batch

processing can be avoided with streams as packets do not have to be temporarily stored in the host memory before being sent to the device.

The packets usually arrive at a Network Interface Card (NIC) on the device and follow a specific datapath. The datapath within edge computing devices consists of the CPU that is connected to the NorthBridge. The RAM and the GPU are connected to the NorthBridge and communicate with the CPU through this channel. The networking interfaces like NIC cards are connected to the SouthBridge. This bridge serves the slower devices like network adapters. The connections are through PCI and have defined bandwidths on different devices. The bandwidth of each connection has to be considered when offloading packets to GPUs. In general cases, RAM has the highest bandwidth on a NorthBridge connection followed by the CPU and then the GPU. This introduces a potential bottleneck in transferring data from the host to the device.

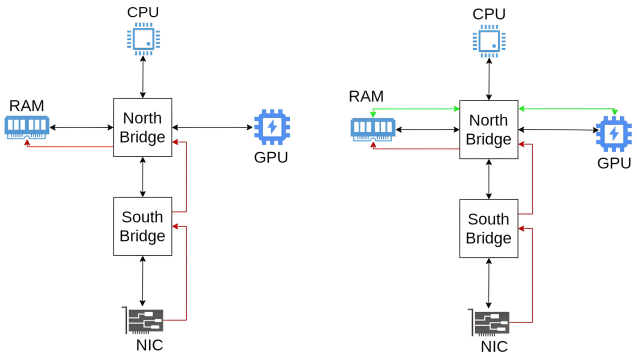

Fig. 2. Network datapath for CPU and GPU based flows.

This drawback can be solved by copying only processing parameters in each packet and keeping the rest in the main memory. The current work uses only a 48 bit Media Access Control (MAC) address on the GPU, whereas the rest of the frame stays in the main memory. This approach leads to about n/k time improvement in memory utilizations where n and k are the packet size and the processing parameter size, respectively.

B. Bloom Filters: Packet Switching

Using the analysis from the first step, the next part of this work deals with using Bloom Filter based lookup on switches for faster packet switching. The lookup in a CPU-only environment and an accelerated GPU implementation is compared in terms of execution speed. The important consideration in this use case is that the simulation is done in user mode due to the availability of CUDA APIs in user mode only. In contrast, most network processing happens in the kernel as it is faster and more efficient.

Bloom Filters are probabilistic compressed set data structures used to accelerate information retrieval. In the context of switching, there are two primary operations that Bloom Filters need to serve, namely mapping interfaces to MAC addresses and searching efficiently in a mapped structure. The mapping phase is compute-intensive but less frequent operation, whereas the lookup is more frequent.

A controllable false positive rate characterizes Bloom Filters. In the current implementation, a known number of items are inserted into the Bloom Filters, and a limiting false positive rate is chosen. Accordingly, the size of the Bloom Filter can be calculated using the following equation:

$$m = \frac{-n * \ln p}{(\ln 2)^2} \tag{1}$$

where:

$n =$ Number of items
$p =$ Required false positive rate

The number of hash functions can be calculated using the following equation:

$$k = \frac{m}{n} * \ln 2 \tag{2}$$

where:

$m =$ Size of the Bloom Filter
$n =$ Number of items

The false positive rate of a Bloom Filter can be calculated from the following equation:

$$p = (1 - e^{-\frac{k*n}{m}})^k \tag{3}$$

where:

$k =$ Number of hashes
$m =$ Size of the Bloom Filter
$n =$ Number of items

The implementation uses an offline software simulation to generate packets. The MAC addresses from these packets are extracted. The MAC addresses are then processed into Bloom Filters with the help of the murmur hash function. The number of hash functions and the size of the Bloom Filter is determined by using the equations stated above. The resulting execution time for this CPU implementation is calculated. For the following setup, Numba, a python extension to CUDA programming, is used. The same packet generator is used, and the MAC addresses are extracted. The parallelization choices are to process the k hash functions independently and process each packet on a separate thread. The CUDA thread organization is implemented such that the threads along the X dimension process packets and the threads along the Y dimension process the hash functions. The thread organization used is [1024, 1024]. For the GPU implementation, the execution time for the kernel routine is considered.

IV. Results and Discussions

This section explains the observed results from the two objectives discussed in this work. The main evaluation criteria are the processing time as observed from both implementations.

A. Experimental Setup

The hardware for running the described algorithms and techniques is an Nvidia Geforce GTX 1650Ti GPU based on the Turing architecture with 1024 CUDA Cores and 4GB of GDDR6 memory. An AMD Ryzen 7 4800H CPU with 8 cores and 16 threads is the CPU used. The system has an Intel Wifi-6 NIC card and an RTL8111/8168/8411 PCI Express Gigabit Ethernet Controller.

B. Performance Considerations

GPU Packet Transfer: The nvprof profiler is used to measure the time taken by both the synchronous single stream copy operation and the asynchronous multi-stream copy operation. The portion of time taken by a blocking copy operation and a stream-based asynchronous copy operation is summarised in Table I and Fig. 3.

TABLE I
COMPARISON OF BLOCKING AND STREAM BASED COPY OPERATIONS.

No. of Packets	Synchronous	Streams
1000	6.72%	3.89%
10000	30.70%	15.29%
100000	53.65%	26.93%
1000000	52.41%	24.50%

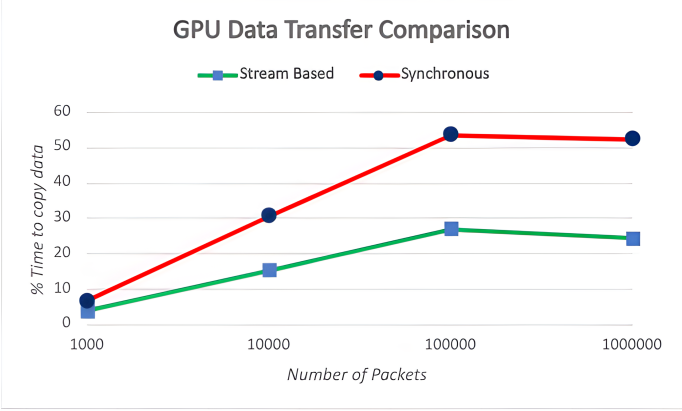

Fig. 3. Results from both copy operations.

A 2x improvement is observed in the time taken for the asynchronous copy operation. This results from the data transfer being done concurrently across multiple streams. The number of streams can be initialized during the startup. Note that the stream creation overhead has to be considered when using asynchronous copy operations. The overhead of creating streams can be ignored in the total time as packet rates are large enough.

Bloom Filter Switching: The packet switching simulation is carried out with a different number of packets. The MAC address is randomly generated and hashed by the murmur hash function. The observed results indicate that for smaller packet numbers, the GPU implementation is slower than the CPU implementation. The main portion of time is being spent in the GPU startup and transfer of data across the host and the device. The results show improvements when the number of packets increases. Execution times have been summarised in Fig. 4.

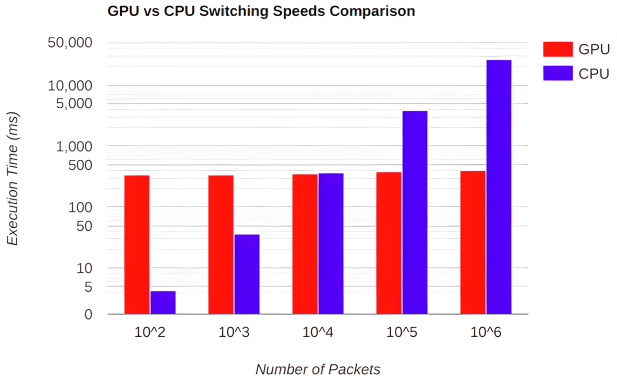

Fig. 4. Execution Times for CPU and GPU.

An observation from the profiled data for the GPU suggests that a significant amount of time is spent in the startup of the GPU routines and data transfer between the host and device. This portion has not been considered in the results above. The mapping phase for switches, being a less frequent operation, can be carried out without significant delays if the copy operation is used a limited number of times.

V. Conclusion and Future Work

The current work explores how high computing GPUs can be used on both the edge and core network devices to improve network processing speeds. The first part of this work shows how using stream-based data transfers can marginally improve data transfer time between GPUs and CPUs which is traditionally the overhead in GPGPU computing. This work is able to get about 2x improvement in the performance with the use of streams. The second part of this work explores how a network switching application can be offloaded into the GPU for faster switching with the help of Bloom Filters. The results show how an increasing number of packets can cause an exponentially increasing processing time on the CPU, whereas the time remains almost constant for a GPU.

The future scope of this work includes an extension in distributed architectures and multiple GPUs. Large networks can use a single high-performance GPU or a collection of multiple centralized GPUs to offload network processing across several devices. Given that CUDA APIs are standardized across GPUs, the techniques described in this work can be used for setting up GPU clusters and switching workload analysis. Since switches are self-learning devices, advanced versions of Bloom Filters like Scalable Bloom Filters and Counting Bloom Filters can be used to provide better results.

REFERENCES

[1] "Cisco Annual Internet Report. Cisco White Paper 2020." https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html, accessed 24 October 2021

[2] Lee, Wai Kong, "High Speed Computation Of Advanced Cryptographic Algorithms On Massively Parallel Architecture," Diss. UTAR, 2018.

[3] Ghorpade, J., Parande, J., Kulkarni, M. and, Bawaskar, "GPGPU Processing in CUDA Architecture," Advanced Computing: An International Journal, 2012, 3(1), pp.105-120.

[4] Babej, Michal, and Pekka Jääskeläinen, "HIPCL: Tool for Porting CUDA Applications to Advanced OpenCL Platforms Through HIP," International Workshop on OpenCL, pp. 1-3. 2020.

[5] Andelfinger, Philipp, Jens Mittag, and Hannes Hartenstein, "GPU-based architectures and their benefit for accurate and efficient wireless network simulations," 19th Annual International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, pp. 421-424, 2011.

[6] T. Suzuki, S. Kim, J. Kani and, J. Terada, "Software Implementation of 10G-EPON Downstream Physical-Layer Processing Adopting CPU-GPU Cooperative Computing for Flexible Access Systems,", IEEE Access, vol. 7, pp. 33888-33897, 2019.

[7] Bhattacharjee, Sangeeta, Satyendra Singh Yadav, and Sarat Kumar Patra, "LTE physical layer implementation using GPU based high performance computing," IEEE International Conference on Advanced Communications, Control and Computing Technologies, pp. 1546-1550, 2014.

[8] S. Lee and W. Ro, "Accelerated Network Coding with Dynamic Stream Decomposition on Graphics Processing Unit", The Computer Journal, vol. 55, no. 1, pp. 21-34, 2010.

[9] Shojania, Hassan, and Baochun Li, "Pushing the envelope: Extreme network coding on the GPU," 29th IEEE International Conference on Distributed Computing Systems, pp. 490-499, 2009.

[10] Pan, Xiaohui, "Efficient network packet signature matching on GPUs," 2nd International Symposium on Instrumentation and Measurement, Sensor Network and Automation (IMSNA), pp. 219-222, 2013.

[11] Čiča, Zoran G., "IP lookup as a critical functionality of packet processors," Telfor Journal 5.1, pp. 8-13, 2013.

[12] Han, K. Jang, K. Park and, S. Moon, "PacketShader: a GPU-accelerated software router," ACM SIGCOMM Computer Communication Review, vol. 40, no. 4, pp. 195-206, 2010.

[13] Zhu, Yuhao, Yangdong Deng, and Yubei Chen, "Hermes: an integrated CPU/GPU microarchitecture for IP routing," 48th Design Automation Conference, pp. 1044-1049, 2011.

[14] Shojania, Hassan, Baochun Li, and Xin Wang, "Nuclei: GPU-accelerated many-core network coding," IEEE INFOCOM 2009, pp. 459-467, 2009.

[15] Silberstein, Mark, Sangman Kim, Seonggu Huh, Xinya Zhang, Yige Hu, Amir Wated, and Emmett Witchel, "GPUnet: Networking abstractions for GPU programs," ACM Transactions on Computer Systems, vol. 34, no. 3, pp. 1-31, 2016.

[16] H. Holm, A. Brodtkorb and M. Sætra, "GPU Computing with Python: Performance, Energy Efficiency and Usability,", Computation, vol. 8, no. 1, p. 4, 2020.

[17] Ezzaki, Fatimazahra, Noreddine Abghour, Amina Elomri, Khalid Moussaid, and M. Rida, "Bloom filter and its variants for the optimization of MapReduce's algorithms: A review," International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech), pp. 1-7, 2020.

[18] Ma, Lin, Roger D. Chamberlain, Jeremy D. Buhler, and Mark A. Franklin, "Bloom filter performance on graphics engines," International Conference on Parallel Processing, pp. 522-531, 2011.

[19] Iacob, Alexandru, Lucian Itu, Lucian Sasu, Florin Moldoveanu, and Constantin Suciu, "Gpu accelerated information retrieval using bloom filters," 19th International Conference on System Theory, Control and Computing (ICSTCC), pp. 872-876, 2015.

[20] Li, Yuchen, Qiwei Zhu, Zheng Lyu, Zhongdong Huang, and Jianling Sun, "DyCuckoo: Dynamic Hash Tables on GPUs," 37th International Conference on Data Engineering (ICDE), pp. 744-755, 2021.

[21] Hayashikawa, Masatoshi, Koji Nakano, Yasuaki Ito, and Ryota Yasudo, "Folded Bloom Filter for High Bandwidth Memory, with GPU Implementations," Seventh International Symposium on Computing and Networking (CANDAR), pp. 18-27, 2019.

[22] P. Reviriego, J. Martinez, D. Larrabeiti and S. Pontarelli, "Cuckoo Filters and Bloom Filters: Comparison and Application to Packet Classification," IEEE Transactions on Network and Service Management, vol. 17, no. 4, pp. 2690-2701, 2020.

[23] H. Byun and H. Lim, "Comparison on Search Failure between Hash Tables and a Functional Bloom Filter," Applied Sciences, vol. 10, no. 15, p. 5218, 2020.

[24] Hyesook Lim, Ji-Hyun Seo and, Yeo-Jin Jung, "High speed IP address lookup architecture using hashing,", IEEE Communications Letters, vol. 7, no. 10, pp. 502-504, 2003.

[25] Günther, Stephan M., Maximilian Riemensberger, and Wolfgang Utschick, "Efficient GF arithmetic for linear network coding using hardware SIMD extensions," 2014 International Symposium on Network Coding (NetCod), pp. 1-6, 2014.

[26] Park, Joon-Sang, Seung Jun Baek, and Kyogu Lee, "A Highly Parallelized Decoder For Random Network Coding Leveraging GPGPU," The Computer Journal, vol 57, no. 2, pp. 233-240, 2014.

[27] Zhang, Shengli, and Soung Chang Liew, "Physical layer network coding with multiple antennas," IEEE Wireless Communication and Networking Conference, IEEE, pp. 1-6, 2010.

[28] Chu, Xiaowen, Kaiyong Zhao, and Mea Wang, "Accelerating Network Coding On Many-Core Gpus And Multi-Core Cpus." Journal Of Communications, vol 4, no. 11, 2009.

[29] Lu, Yang et al. "Optical Label Switching Based On Manchester Code + NRZ Modulation," Optical Fiber Technology, vol 61, p. 102387, 2021.

[30] Greenberg, Shlomo et al. "Packet Classification Using GPU And One-Level Entropy-Based Hashing," IEEE Access, vol 8, pp. 80610-80623, 2020.

[31] Chinchilla R, Hoag J, Koonce D, Kruse H, Ostermann S, Wang YC, "The trafgen traffic generator," Telecommunication Systems, 2002.