gR: A GPU-based Router

Priya Sundaresan, Sripriya Venkateshprasad, Yamini Muralitharan and Ranjani Parthasarathi
Department of Information Science and Technology
Anna University
Chennai, India

Abstract—**With the growing internet traffic and complexity of packet processing task, the throughput of routers is affected. Also modern routers need to provide additional services like security, QOS which further adds to the complexity. These issues can be addressed with the massive parallel computing capability of graphic processors. In this paper, we offload two of the most computationally intensive tasks namely lookup and two-key IPSec to CUDA-enabled GPU. The execution time is reduced by exploiting the inherent parallelism in packet processing tasks. Finally, we have analyzed the performance variations and have found that modern graphic cards can be effectively used to speed up router operations.**

I. INTRODUCTION

A substantial increase of traffic in today's Internet has posed a great challenge for complex packet processing tasks. Commercial hardware like FPGA, NPU (Network Processing Unit), TCAM are proposed to meet the demand. But these solutions may not be cost-effective and easy to manage because they require specific hardware and configuration.

This work approaches the problem differently by offloading network packet processing tasks to GPGPUs. GPU's massive parallel processing power has been used in various application domains like cryptography, computer vision, weather forecasting etc. Network processing is another domain large scope for parallelism as it has large amount of packets to be processed with least or no dependency between them. Further, the scalability and easy programmability of modern GPUs can help in development of high-speed cost-effective router.

This paper aims at exploring effective implementation of network processing task on a CUDA-enabled GPU. We focus on accelerating two operations of router: IP lookup which is considered as representative task of normal router operation and two-key IPSec that is essential for deep packet inspection on edge-routers.

This paper is organized as follows: Section 2 discusses the previous work carried out. Section 3 gives the implementation details of lookup and IPSec. Section 4 delivers the performance analysis and Section 5 concludes with scope for future work on GPUs.

II. RELATED WORK

Many works have been carried out to improve the performance of lookup and IPSec. The primary technique used in lookup is Longest Prefix Match (LPM) and are usually implemented with trie data structure. In case of IPSec, cryptographic algorithms form the basis for providing security.

A. Lookup

Several software and hardware related solutions have been proposed for LPM algorithms for efficient IP lookup. One of the idea proposed by Waldvogel [16] is binary search on prefix length using hash tables. Dharmapurikar et al. [3] proposed the idea of using bloom filters for lookup. The destination address is hashed to find which bloom filter it belongs to and the corresponding prefix entries are traversed to find the next hop. Other techniques for finding the longest matching prefix are path compression [10], prefix expansion [15], level compression [11] to reduce the worst case memory access times per lookup. Path compression technique for lookup involves reducing the skip count (number of non-terminating nodes, traversed) while traversing the trie for finding a match. Prefix expansion technique operates by extracting multiple bits at a time and using them as an index of array pointers to traverse the child nodes. In Level Compression tries, whenever a trie node does not have terminal nodes for the next t levels, the fan-out trie is compressed into a single array of size 2^t and t bits are used to index into this array of pointers to nodes. All these trie-based scheme suffers from disadvantage of choosing a child node at each level of traversal (maximal dependency on parent node). This makes it difficult to implement these schemes in a parallelized manner. Also, these trie-based algorithms were particularly designed for 32-bit v4 addresses and don't scale to 128-bit v6 addresses.

Several hardware related solutions have also been proposed like CAMs, FPGAs, GPUs. CAM based lookup schemes proposed include prefix segregation[9],single-match technique[6], load balancing multi-chip technique [8] and exploitation of inherent prefix properties [12]. Hoang Le et al. [4] and Zoran Chicha et al. [17] discuss how FPGAs are used in providing better lookup performance. Generally CAMs are much slower, more expensive; several orders of magnitude smaller than conventional memory and FPGAs are expensive and sophisticated. So GPUs are better suited for cost-effective designing of PC-based routers. Jin Zhao et al., Sangjin Han et al. and Shuai Mu et al. [7,13,14] discuss several lookup schemes done using GPU. Some of these techniques [7] are not memory efficient. To achieve better memory usage and performance we implement bloom filter technique (minimum dependency, better parallelization) on CUDA enabled GPU.

B. IPSec

Several solutions have been proposed for the implementation of complex cryptographic algorithms like AES and DES on parallel processing architectures like GPU and FGPA [2]. But implementing these algorithms itself in parallel as in the context of two-key IPSec, is still an open area of research. To our knowledge, this is the first paper to discuss two-key IPSec on GPU

III. Proposed work

In this work, we use NVidia GeForce GT220. It has 48 cores with compute capability 1.2. Appropriate algorithms have been chosen to exploit parallel computing ability of CUDA GPU.

A. *IP lookup*

Bloom filters are efficient data structures that can represent large amount of data items with relatively less memory space. The storage and search operations can be done in O(1). Hence, bloom filters are good candidates for lookup. In this work we use bloom filters and perform parallel lookup on prefixes to find Longest Prefix Match (LPM). The bloom filter construction and parallelization of lookup on CUDA are discussed below.

1) Bloom Filter Construction: The routing table entries are grouped by prefix length and bloom filters are constructed for each length. The optimal size of bloom filter and the number of hash functions required are computed using the following formulae,

$$M = -2 * N * \ln(P) \tag{1}$$

$$K = 0.7 * M / N \tag{2}$$

where,

N – Maximum number of data items

P – Allowed false positive rate

M – Bloom Vector size

K – Optimal number of hash functions

Instead of using independent hash functions, which are computationally expensive, we use single hash function to simulate hash values without significance loss in asymptotic false positive probability. The math behind this technique, as described in [1], is

$$g_i(x) = h_1(x) + i * h_2(x) \tag{3}$$

where, h_1 and h_2 are the initial hash values and $0 \leq i < K$. These hash values are used as indices to set bloom vector. Since only two values (0 and 1) are used to represent data, we adopt bitwise storage for effective memory utilization.

2) Parallelization on CUDA: Here, we have briefed about how parallelization can be done on CUDA. Important considerations to be made on mapping to CUDA are as follows: Thread organization and Memory organization.

- Thread organization - CUDA architecture is organized as a collection of grids, blocks and threads. A grid is a collection of blocks. A block is a collection of threads. During implementation, a decision has to be made on how these are organized to obtain peak performance.
- Memory Organization - CUDA memory architecture has different types of memory like constant, global, shared etc. During implementation, suitable memory has to be chosen for storing the data, by considering the pros and cons of each memory.

Execution of a CUDA program involves launching kernels. A kernel is a piece of code which is callable from host and executed on device.

In lookup, parallelism is be achieved at data-level and task-level. The incoming packets are handled in parallel by the y-dimension threads, which accounts to data-level parallelism. For each packet, the x-dimension decides the bloom filter to be queried for prefix match, thus accounting to task-level parallelism. This facilitates communication between threads in same warp to obtain LPM. The next hop is obtained by traversing the entries in table corresponding to longest prefix length. Fig. 1 shows block launch with 2-D threads.

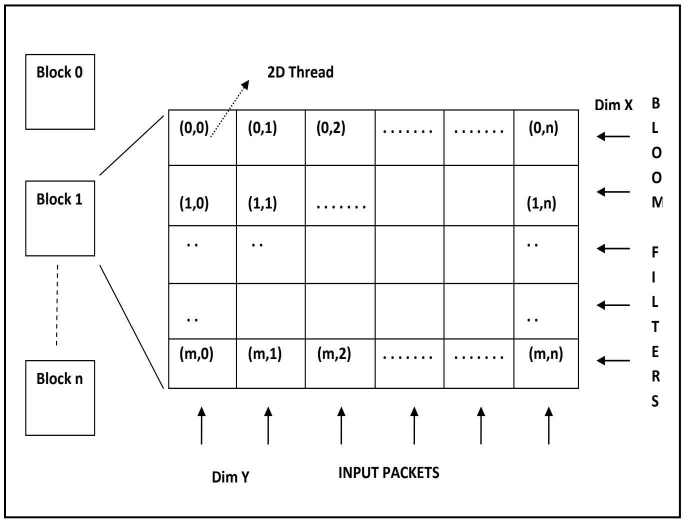

Figure 1. Block launch for lookup

Memory optimization: Bloom vectors are loaded in read-only constant memory for fast access. Further, we use bitwise storage of bloom vector as mentioned above, to reduce memory usage.

B. *Two-key IPSec*

IPSec aims at providing security and privacy by encrypting packets. It employs several cryptographic algorithms for this purpose. In the scenario considered here, we need to perform partial decryption of two-key IPSec encrypted packets to allow deep packet inspection. We restrict ourselves to the implementation of AES and DES decryption algorithms.

AES and DES operating on large blocks of data are computationally intensive and largely byte-parallel. At a high level, these decryption algorithms take in a 128-bit key or 64-bit key in case of AES and DES respectively and a variable length message. They split the message into 128-bit blocks/64-bit blocks, and perform the decryption steps on these bit blocks independently.

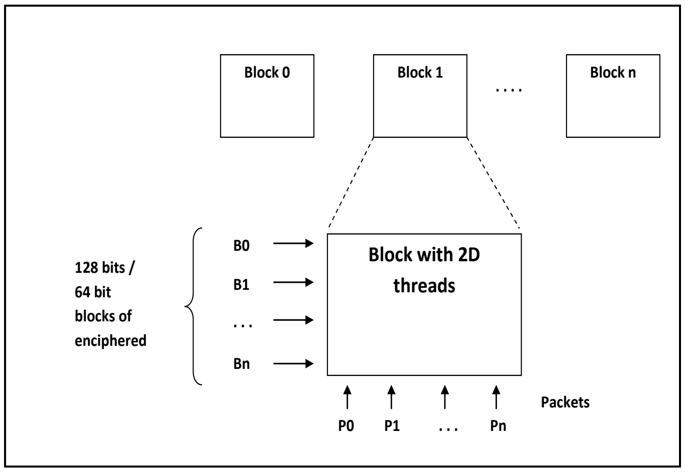

Figure 2. View of block for two-key IPSec

This naturally leads to a highly parallel GPU implementation. The GPU threads perform the AES and DES decryption functions in parallel. This is important in the context of IPSec, as different packets are encrypted with different algorithms. Each thread works on a subset of the packet data (128/64 bit blocks), so there are no dependencies between threads. Parallelization is depicted in Fig. 2, where packets are handled in parallel, with each y-dimension thread handling a packet and its corresponding decryption algorithm. The x-dimension threads handles byte blocks within a packet in parallel, thus providing algorithm-level parallelization.

The Key expansion process of both these algorithms is inherently serial. So, the keys are pre-computed and are then applied to the threads.

To optimize global memory access, it is necessary to coalesce data accesses together to allow the GPU to perform wide memory reads. Since all the threads access the data from global memory before processing the data, it is easy to order the accesses such that the most efficient usage or the memory bus is achieved. Another optimization that is useful is the use of constant memory. Both AES and DES use several tables to perform lookups. Since these tables are constant and common between threads, we can load them into the constant memory of the GPU. This memory is cache-backed and the cache can accommodate all of the table data. Once the data is in the cache, there is little penalty in accessing it again

IV. PERFORMANCE

The implementation is done on Nvidia GT220 graphics card which has 48 cores. We have varied the number of threads and blocks and have evaluated the performance. By varying these parameters, the best combination for optimal performance of these operations is identified. The experiment is conducted with 10K router entries obtained from IPV4 BGP table. This work is scalable to IPV6 addresses also. The time taken for different block-thread combinations for 1, 20,000 packets is shown in Table 1.

From the speed-up graph shown in Fig. 3, it can be seen that the peak value is obtained for 12-2048 blocks-thread combination. In case of IPSec, it can be inferred that the peak performance is achieved for 6 blocks and 256 threads as shown in Fig.4

TABLE I. LOOKUP PERFORMANCE FOR 1, 20,000 PACKETS

Block	2D-threads	Time(ms)	Speed-up
6	1024	2.789	1
	2048	2.006	1.39
	4096	2.767	1.01
12	1024	3.614	0.77
	2048	1.123	2.48
	4096	1.346	2.07
18	1024	3.155	0.88
	2048	2.267	1.23
	4096	2.112	1.32

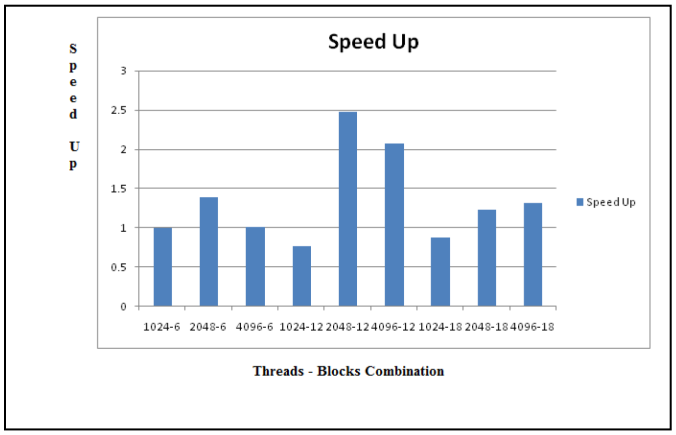

Figure 3. Speed-Up for Lookup Performance

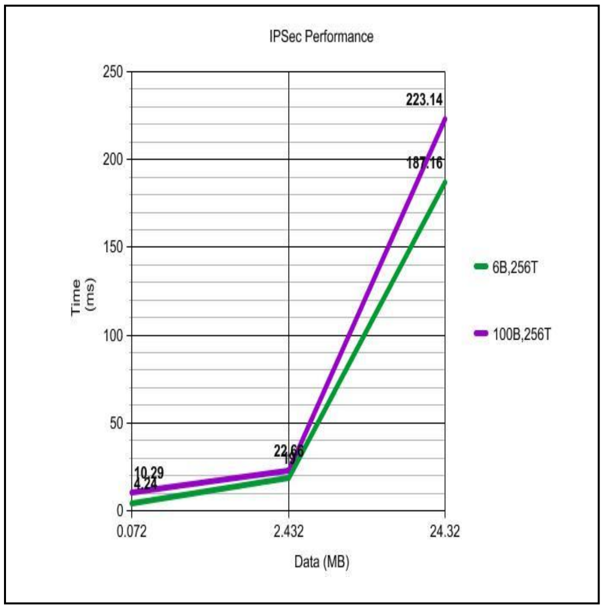

Figure 4. IPSec Performance

It is to be noted from Fig. 4 that though the increase in the number of blocks from 6 to 100 leads to increase in the number of packets being processed in parallel, the overall execution time increases as opposed to an expected decrease. This can be supported by the fact that, increase in the number of blocks poses block scheduling overhead thus having adverse effects on the execution time.

A close look on both graphs indicates that optimal execution time is achieved at a point where the resources are utilized effectively.

The lookup performance is compared against AMD athlon with 512M DDR33[5].Incase of AMD athlon, for maximum of 753 entries and 3812.70KB the time taken for lookup is 0.744 sec. But our performance on GPU yields 0.0011 sec for 10K entries and 3988KB of data. From the comparison discussed above, it can be seen that GPU's parallel computing capability results in better performance. So, GPUs can be used for designing software routers with better throughput.

V. CONCLUSION

This paper supports the idea that GPUs can be used for designing cost-effective high-speed PC-based routers. The work can be extended for other router operations. Our work was done using a single GPU. Future work can extend this to multiple GPUs. The issues with current GPUs are being addressed by new and upcoming architectures like Nvidia's Fermi. More focus can be given for deploying network operations on these latest GPUs.

REFERENCES

[1] A. Kirsch, M. Mitzenmacher, "Less hashing, same performance: Building a better Bloom filter," Random Struct. Algorithms, col. 33, pp. 187-218, Sept.2008.

[2] C. Fiorese, C. Budak, "AES on GPU: a CUDA Implementation," In CHES, pp 209–226, 2007.

[3] S. Dharmapurikar , P. Krishnamurthy, D. E. Taylor, "Longest prefix matching using bloom filters," Networking, IEEE/ACM Transactions, vol.14, no.2, pp. 397- 409, April 2006.

[4] H. Le, W. Jiang, and V. K. Prasanna, "A SRAM-based Architecture for Trie-based IP Lookup Using FPGA," Proc. Sixteenth International Symposium on Field-Programmable Custom Computing Machines (FCCM '08), IEEE Computer Society, Washington, DC, USA, 2008, pp. 33-42.

[5] H. Ma, X. Deng, Y. Ma, Z. Li, "Divide-and-Conquer: A Scheme for IPv6 Address Longest Prefix Matching," in IEEE conference publication, June-2006.

[6] Jinsoo Kim and Junghwan Kim, "An efficient IP lookup architecture with fast update using single-match TCAMs," Proc. Sixth International conference on Wired/wireless internet communications (WWIC'08), Berlin, Heidelberg, 2008, pp. 104-114.

[7] J. Zhao, X. Zhang, X. Wang, and X. Xue, "Achieving O(1) IP lookup on GPU-based software routers," SIGCOMM Comput. Commun. Rev. 40, August 2010, pp. 429-430.

[8] K. Zheng, C. Hu, H. Lu, B. Liu, "An Ultra High Throughput and Power Efficient TCAM-Based IP Lookup Engine," Tsinghua University, Beijing, China, IEEE INFOCOM 2004.

[9] M. J. Akhbarizadeh, M. Nourani, C. D. Cantrell, "Prefix Segregation Scheme For a TCAM-Based IP Forwarding Engine," University of Texas,Dallas,IEEE 2005.

[10] D.R. Morrison, "PATRICIA--Practical Algorithm to Retrieve Information Coded in Alphanumeric," J. ACM, vol. 15, no.4, pp. 514-534,Oct.1968.

[11] S. Nilsson and G. Karlsson, "IP-Address Lookup Using LC-Tries," IEEE J. SAC, vol. 17, no. 6, June 1999, pp. 1083–1092.

[12] V.C. Ravikumar and R. N. Mahapatra ," TCAM Architecture For IP Lookup Using Prefix Properties," Texas A&M University,IEEE 2004.

[13] S. Han, K. Jang, K. Park, S. Moon,"PacketShader: A GPU-Accelerated Software Router," SIGCOMM'10, August 30–September 3, 2010, New Delhi, India.

[14] S. Mu, X. Zhang, N. Zhang, J. Lu, Y. S. Deng, and S. Zhang, "IP routing processing with graphic processors," Proc. Conf. on Design, Automation and Test in Europe (DATE '10), European Design and Automation Association, 3001 Leuven, Belgium, 2010, pp. 93-98.

[15] V. Srinivasan. and G. Varghese, "Fast Address Lookups Using Controlled Prefix Expansion," ACM Transactions on Computer Systems, Vol. 17,No. 1, February 1999, pp. 1–40.

[16] M. Waldvogel, "Fast longest prefix matching: Algorithms, analysis and applications," Ph. D. Thesis, Swiss Federal Institute of Technology,Zurich, 2000.

[17] Z. Chicha, L. Milinkovic, A. Smiljanic , "High Performance Switching and Routing,FPGA Implementation of Lookup Algorithms," Belgrade University, 12th International Conference,IEEE 2011.