Scalable Packet Classification via GPU Metaprogramming

Kang Kang and Yangdong Steve Deng
kk06.thu@gmail.com and yangdong.deng@gmail.com
Institute of Microelectronics, Tsinghua University

***Abstract*—Packet classification has been a fundamental processing pattern of modern networking devices. Today's high-performance routers use specialized hardware for packet classification, but such solutions suffer from prohibitive cost, high power consumption, and poor extensibility. On the other hand, software-based routers offer the best flexibility, but could only deliver limited performance (<10Gbps). Recently, graphics processing units (GPUs) have been proved to be an efficient accelerator for software routers. In this work, we propose a GPU-based linear search framework for packet classification. The core of our framework is a metaprogramming technique that dramatically enhances the execution efficiency. Experimental results prove that our solution could outperform a CPU-based solution by a factor of 17, in terms of classification throughput. Our technique is scalable to large rule sets consisting of over 50K rules and thus provides a solid foundation for future applications of packet context inspection..**

***Keywords*—Packet Classification; Software Router; GPU; CUDA; Metaprogramming**

I. INTRODUCTION

Today our world is connected by the Internet. Among numerous network devices, Internet routers play an essential role by serving as the backbone. A router is responsible for delivering packets between neighboring networks in a timely manner. Current Internet routers are "flow-aware", which means they classifies all incoming packets into flows according to a set of predefined rules, before any further processing can be done accordingly. Such process known as packet classification is a key step in providing quality of service (QoS) control and other advanced router functionalities.

Traditional routers depend on custom hardware to attain high processing speed [1]. Specifically, ternary content-addressable memory (TCAM) is used for packet classification. Besides being expensive, such solutions cannot fully adapt to fast changing network services and diversified usage cases. On the contrary, software routers have been attracting significant research efforts recently (e.g. [2] [3]) due to their extensibility and customizability. Software routers are cost-efficient and flexible because they are built with commodity hardware.

However, despite their superior flexibility, software routers can hardly deliver the performance level required by fast-rising Internet traffic. The fastest existing CPU-based software router [3] reports a forwarding throughput of near 10Gbps, whereas carrier-grade devices have to reach 40Gbps and scale to even as high as 92Tbps [4]. The relatively low performance has significantly limited the application of software routers.

A new trend in high-performance computing is to perform general purpose computing with graphic processing units (GPUs) [5]. A couple of recent works used GPUs to implement software routers and observed significant boost in speed [6] [7]. However, the pioneering work reported in [6] focused on full system implementation and did not fully investigate the packet classification algorithms. PacketShader [7] has implemented the OpenFlow switch protocol [8] which is a specialized packet classification standard designed for research of new protocols. OpenFlow differs fundamentally from general packet classification schemes due to its dedicated purpose.

In this work, we investigate efficient data-parallel algorithms for packet classification, following the spirit of previous works on GPU-based software routers [6] [7]. A key observation is that many algorithms, although having low theoretical complexity, may scale poorly with large rule sets, and fail to meet performance constraints under future large rule sets [9]. We address the scalability problem by avoiding overly complicated algorithm and instead focusing on efficient parallelization. Adopting a novel metaprogramming approach, we propose a GPU-based linear search scheme for packet classification. Experimental results prove that it significantly outperforms existing software solutions, even with very large rule sets.

II. PRELIMINARIES FOR PACKET CLASSIFICATION

A. *Problem formulation*

Packet classification works on the header of input packets. The problem can be defined as follows [10]:

Given a set of N rules, where each rule has d components, and each component is a criterion on a specific field of the packet header. A packet P matches rule R if every one of the d fields in its header meets the corresponding criterion in R. Since a packet may match more than one rule, priority is assigned to rules in order to break ties. In summary, packet classification is the process of finding the highest-priority matching rule for every incoming packet.

In practice, packet classification involves the following header fields: source and destination IP addresses (SIP/DIP), port numbers (SP/DP), and protocol ID (PROT). The 5-tuple <SIP, DIP, SP, DP, PROT> forms a 104-bit vector which fully captures the characteristics of a packet header.

B. *Current solutions*

Most hardware routers implement packet classification with the ternary content-addressable memory (TCAM) [11], which performs packet classification as a single-step lookup. In spite of the high processing speed, TCAMs are hindered by four major deficiencies [11]: 1) high cost, 2) storage inefficiency, 3) high power consumption, and 4) limited scalability for large rule sets. On the other hand, software packet classification systems perform the rule matching process on general-purpose processors (e.g. BPF+ [12] and Click [2]). Such systems are usually feature-rich and highly configurable, but weak in performance.

A large number of software packet classification algorithms have been proposed in recent years. Taylor's taxonomy [11] categorizes these algorithms into four classes, namely the exhaustive search, decision tree, hash table, and decomposition. The simplest and most fundamental method of classifying packets is an exhaustive search over the entire rule set [11]. It is also known as linear search when performed in a sequential manner. Exhaustive search does not attempt to reduce redundancies in the rule set at all. Therefore, it incurs considerable computational overhead. It is frequently used as a component within more sophisticated algorithms. Note that exhaustive search is readily parallelizable.

Decision tree and hash table algorithms both aim to divide original rule set into subsets, in order to cut down the number of rules that need to be iterated. We may view k-dimensional packet classification as a problem of finding the highest-priority hyper cube that covers a given point in a k-dimensional classification space, where each rule corresponds to a c-dimensional ($1 \le c \le k$) hyper cube, and a packet equals to a point in the space [13]. From this point of view, both decision tree and hash table methods seek to cut the classification space into 2^p subspaces using a set of p hyper planes, so that there are only a few rules in each region. The major difference between the two lies in the underlying data structure.

C. *Performance challenges*

For software packet classification solutions, performance metrics reduce into two categories: speed, and scalability.

- **Speed** Classification speed is usually the major challenge for software packet classification. In the worst case of uniformly minimum-sized packets, a throughput of 25.5Mpps is required to support the OC-192 line speed of 10Gbps [14].
- **Scalability** In addition to classification speed, deficiencies in other metrics including storage size and preparation time may become stumbling blocks for extensibility. For instance, some classic algorithms are reported to cease to work even with mid-sized rule sets due to unrealistic storage consumption and preparation time (e.g. over 4GB of memory and 24 hours of time [9]).

III. GPU Computing For Packet Classification

A. *GPU architecture and programming model*

We use the NVIDIA GT200 GPU to exemplify modern GPU architectures. The GPU chip consists of 240 scalar processors (SPs), evenly distributed into 30 streaming multiprocessors (SMs). A single SP has its own execution hardware, but no instruction fetch and decoding capabilities. A SM would fetch instructions and schedule their execution on its 8 internal SPs.

Programming for the NVIDIA GPU follows the Compute Unified Device Architecture (CUDA) [15] programming model. A GPU application could launch tens of thousands of threads, which execute the same program but work on different data sets. Multiple threads are organized into thread blocks in a 1-D, 2-D, or 3-D manner to match the problem structure. During execution, every 32 threads known as a "warp" would follow an identical instruction schedule, i.e., a SIMD execution style [15]. If threads in a group take different branches of a program, all possible program execution paths have to be traversed. The program data would be stored on the graphics memory, which is located off the GPU chip, but on the graphics card. The amount of cache on a GPU chip is very limited; long memory latency has to be hidden by 1) coalescing memory requests to neighboring addresses into a single request, and 2) switching among threads. On the other hand, the program instruction is cached. Later we will show the difference in caching data and instruction has important implications for GPU-based packet classification.

B. *GPU-accelerated network processing*

We implemented packet classification algorithms on a GPU-accelerated software router as illustrated in Fig. 1. The Internet traffic is conducted through two 10Gbps network interface cards (NICs). The packets would first be transferred from NICs to main memory via DMA operations. Before processed by GPU, packets are transported to GPU memory through the PCIe bus attached to the North Bridge. The routing operations are orchestrated by the CPU, while the majority of packet processing workload is offloaded to GPU to utilize its massive parallel computational resources.

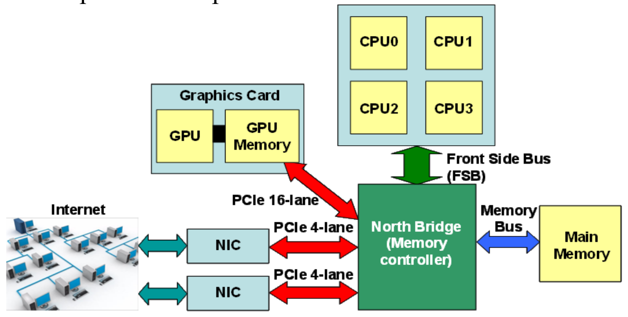


Fig. 1. GPU-based software router.

Regarding GPU-based packet classification implementations, the hash-table-based and exhaustive search algorithms are good candidates due to their relatively regular data flow. Decision tree algorithms, however, are unsuitable since many of their inherent characteristics are contradictory to the basic principles of GPU computing. Especially, traversing of a tree structure leads to constantly divergent program execution paths, and thence considerable overhead.

C. *GPU packet classification using DBS algorithm*

We investigate the discrete bit selection (DBS) [16] as a representative of hash-based algorithms. The algorithm is selected for its simplicity and fair performance. In addition, data preparation in DBS is relatively fast; for many algorithms, preparation time can become the bottleneck on large rule sets.

Classification using DBS algorithm works as follows. A certain number (16 in our case) of "effective bits" are consecutively chosen from the 104-bit vector formed by concatenating all 5 fields. The effective bits then serve as the hash key mask in filling and probing the hash table.

We implemented DBS on both GPU and CPU. Preparation is conducted on CPU in both versions. GPU threads are organized into 120 blocks, each consisting of 128 threads. One thread is responsible for the processing of one packet. Therefore, 14,336 packets are processed as a batch during each kernel launch.

An interesting observation obtained from DBS, that the partitioning of rule sets is limited to a certain granularity, implies a performance bound for more general algorithm categories than only DBS or hash-based algorithms. The limit is due to the overlapping of rules in the classification space. Consider the situation where one packet may simultaneously match a group of m rules: all m rules must be iterated through in order to find the highest-priority one. This poses a worst-case bound on the number of lookups.

We measured the minimum worst-case numbers of lookups, i.e., the largest number of rules that overlap and thence are impossible to be further divided. Fig. 2. Shows the results: in most cases it resolves to a few dozen lookups, but lower bound for worst-case number of lookups can be as high as 167 for some rule sets. This has a larger impact on GPU than on CPU on classification performance. The reasons are twofold. First, GPU undergoes much larger overhead in memory operations. Second, there is no early termination mechanism available for GPU computing since all threads in a warp must take the same execution path; a GPU's SMs must wait until the "slowest" thread finishes before taking in another warp. The worst case behavior on CPU then becomes the average case behavior on GPU.

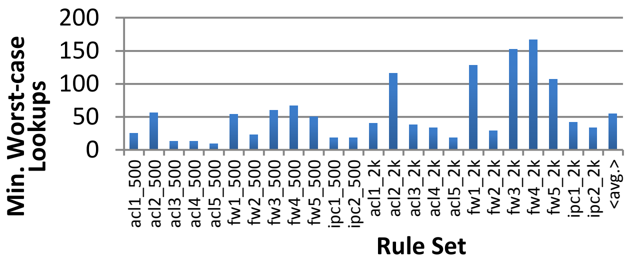


Fig. 2. Limitation on rule set partitioning.

D. *GPU packet classification using linear search*

The analysis in the previous section suggests that the hash-based classification algorithms incur inherent hurdles for an efficient GPU implementation. In attempt to entirely circumvent these difficulties, we developed a linear search scheme for software packet classification. The proposed linear search scheme is radically different from common approaches in that no memory accesses are needed to fetch rules.

The key to eliminating memory access is the adoption of a metaprogramming technique. Metaprogramming refers to the practice of generating source or executable code dynamically from a more abstracted program, i.e., a metaprogram. The heterogeneous nature of GPU computing system is actually very amendable to metaprogramming, because the CPU is readily available for code generation and flow control, whereas the CUDA Driver API allows dynamical compiling, uploading, and executing of GPU kernel code in runtime. Our prototype system performs packet classification in the following steps:

1) Translate each rule into a fragment of C code that checks if a packet matches. The rules are now embedded as compilation-time constants.
2) Form a series of kernels with the code fragments, compile them and upload binaries to the graphics card.
3) Load packet header data into GPU memory, run the kernels, then transfer classification results back to CPU.

This technique eradicates fetching of rules from memory; instead, the rule set is compiled as an integral part of program code, which can be efficiently broadcast to all CUDA cores on a GPU through the instruction issuing mechanism. Moreover, the memory latency for instruction binary can be automatically and effectively hidden by GPU's instruction cache.

The same thread block configuration as in the GPU version of DBS is used. To minimize compilation time, we create a new kernel for every 200 rule entries. A CPU version of the same algorithm is implemented for comparison.

IV. Performance Evaluation

A. *Dataset and test environment*

Our test data are generated using the ClassBench tool [17]. For each of the 12 parameter sets provided, we synthesize two rule sets with N = 500 and 2000, respectively. To evaluate scalability, an exceptionally large rule set with N = 50,000 is generated for the fw4 parameter set. For each rule set, a sequence of 100,000 packet header traces is generated.

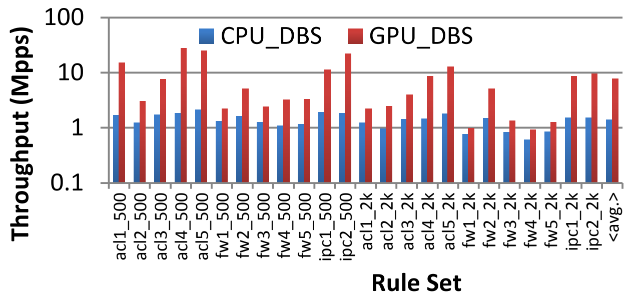
We conduct the tests on a PC with a 2.5GHz Core 2 Q8300 CPU, 4GB memory, and an NVIDIA GTX280 graphics card which features a GT200 GPU with 240 CUDA cores, and 1GB graphics memory. All CPU programs are single-threaded.

B. *DBS algorithm*

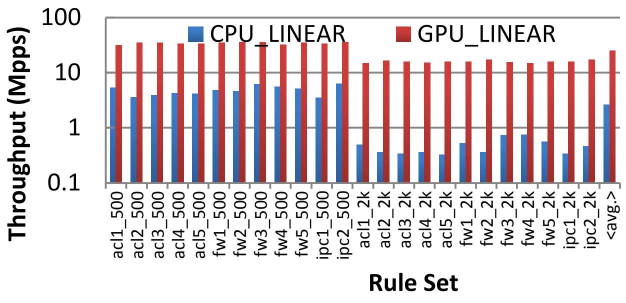
- **Throughput** (Fig. 3A) DBS on GPU shows considerable fluctuation in performance, whereas its CPU counterpart maintains a steady throughput of 1-2Mpps. This proves the previous analysis that the average case performance on GPU corresponds to the worst case behavior on CPU. The GPU implementation delivers an average throughput of 10.7Mpps and 4.8Mpps with rule numbers of 500 and 2000, respectively. On average, it is 4.6 times faster than the CPU equivalent.
- **Scalability** (Fig. 3C) With fw4 being the most complicated rule set in our test, both preparation and classification phases are slow. It is worth noting that preparation time increases faster than classification time.

C. *Linear search*

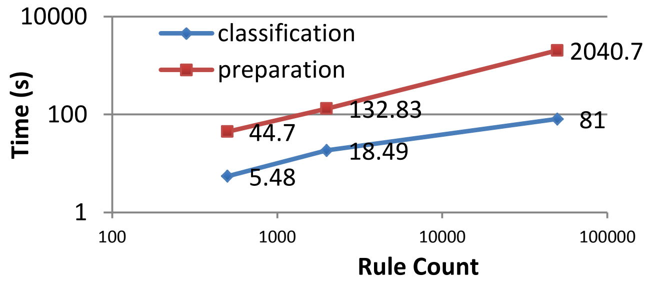
- **Throughput** (Fig. 3B) Linear search is insensitive to the characteristics of rule sets. This insensitivity is more evident on the GPU version, as its threading model forbids early-termination optimizations. Compared with the CPU



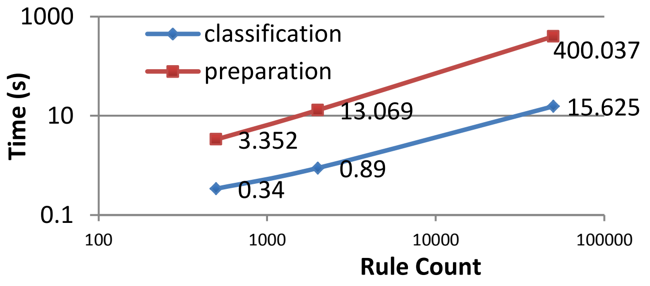
(A) Classification throughput using DBS algorithm.



(B) Classification throughput using linear search.



(C) Scalability of GPU DBS algorithm (fw4 rule set).



(D) Scalability of GPU linear search (fw4 rule set).

Fig. 3. Performance comparison of DBS algorithm and linear search.

implementation, on average the GPU version is 6.2 times faster with N=500, and 32.9 times faster with N=2000.

- **Scalability** (Fig. 3D) Both preparation and classification times grows linearly with rule set size. And time consumption remains in reasonable range for N as large as 50,000.

D. *Summary*

Our GPU metaprogramming linear search scheme is extremely efficient. The CUDA Profiler reports 109% instruction throughput and near-zero global memory accesses. Such high run-time efficiency has proven to effectively amend the algorithmic overhead: GPU linear search can provide 17 times higher throughput than DBS algorithm on CPU, or 8.5 times higher than its own CPU counterpart.

V. Conclusion

In this work, we explore GPU-based acceleration techniques for packet classification, which is a key module in routers and a basic pattern of network processing. We propose a GPU-based packet classification solution using a linear search algorithm. Our scheme exploits a metaprogramming technique by compiling the rules into instructions so as to avoid the expensive latency of memory accesses. The prototype implementation offers significantly higher classification throughput than its CPU counterpart, while at the same time maintains good scalability even when the size of rule set reaches 50,000. We also demonstrated that metaprogramming can be a powerful tool for GPU computing.

References

[1] H. J. Chao and B. Liu, *High Performance Switches and Routers*. Wiley-IEEE Press, 2007.

[2] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The Click modular router," *ACM Transactions on Computer Systems (TOCS)*, vol. 18, no. 3, pp. 263-297, Aug. 2000.

[3] M. Dobrescu, et al., "RouteBricks: Exploiting parallelism to scale software routers," in *Proc. ACM SIGOPS*, 2009, pp. 15-28.

[4] W. Eatherton, "The push of network processing to the top of pyramid," Keynote address at Symposium on Architectures for Networking and Communications Systems, 2005.

[5] J. D. Owens, et al., "A survey of general-purpose computation on graphics hardware," *Computer Graphics Forum*, vol. 26, no. 1, pp. 80-113, Mar. 2007.

[6] S. Mu, et al., "IP routing processing with graphic processors," in *Proc. DATE*, 2010, pp. 93-98.

[7] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader: a GPU-accelerated software router," in *Proc. ACM SIGCOMM*, 2010, pp. 195-206.

[8] N. McKeown, et al., "OpenFlow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69-74, Apr. 2008.

[9] Y. Qi, L. Xu, B. Yang, Y. Xue, and J. Li, "Packet classification algorithms: from theory to practice," in *Proc. IEEE INFOCOM*, 2009.

[10] P. Gupta and N. Mckeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24-32, Mar. 2001.

[11] D. E. Taylor, "Survey and taxonomy of packet classification techniques," *ACM Computing Surveys*, vol. 37, no. 3, pp. 238-275, 2005.

[12] A. Begel, S. McCanne, and S. L. Graham, "BPF+: exploiting global data-flow optimization in a generalized packet filter architecture," *ACM SIGCOMM Computer Communication Review*, vol. 29, no. 4, pp. 123-134, Oct. 1999.

[13] S. Singh, F. Baboescu, G. Varghese, and J. Wang, "Packet classification using multidimensional cutting," in *Proc. ACM SIGCOMM*, 2003, pp. 213-224.

[14] D. Liu, B. Hua, X. Hu, and X. Tang, "High-performance packet classification algorithm for many-core and multithreaded network processor," in *Proc. ICCASES*, 2006, pp. 334-344.

[15] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *ACM Queue*, vol. 6, no. 2, pp. 40-53, 2008.

[16] B. Yang, X. Wang, Y. Xue, and J. Li, "DBS: a bit-level heuristic packet classification algorithm for high speed network," in *Proc. ICPADS*, 2009, pp. 260-267.

[17] D. E. Taylor and J. S. Turner, "ClassBench: a packet classification benchmark," *IEEE/ACM Trans. Netw.*, vol. 15, no. 3, pp. 499-511, 2007.