P4GPU: Accelerate Packet Processing of a P4 Program with a CPU-GPU Heterogeneous Architecture

Peilong Li Yan Luo

Department of Electrical and Computer Engineering,
University of Massachusetts Lowell

Peilong_Li@student.uml.edu, Yan_Luo@uml.edu

ABSTRACT

The P4 language is an emerging domain-specific language for describing the data plane processing at a network device. P4 has been mapped to a wide range of forwarding devices including NPU, programmable NICs and FPGAs, except for General Purpose Graphics Processing Unit (GPGPU) which is a salient parallel architecture for processing network flows. In this work, we design a heterogeneous architecture with both CPU and GPU as a P4 programming target, and present a toolset to map a P4 program onto the proposed architecture. Our evaluation reveals that a P4 program can render promising performance on such architecture by parallelizing its “match+action” engine with the GPGPU accelerator. The experiment results show that the auto-configured GPU kernels achieve scalable lookup and classification speeds: the prototype system can reach up to 580 Gbps for IP lookups (64-byte packets) and 60 million classifications per second for 4k firewall rules, respectively.

Keywords

GPU; Heterogeneous; Packet Processing; P4

1. INTRODUCTION

As a domain specific language for network processing, P4 [1] enhances the programmability of software defined network by providing high-level abstractions and easy-to-use semantics to implement network data plane functions, however, line rate may be compromised if without an efficient hardware target. The whole P4 community is dedicated in extending P4 to a wide range of forwarding devices including CPUs, NPUs, programmable NICs and FPGAs, etc.. GPGPUs, as an SIMD computing architecture, have shown scalable accelerated performance on network applications [3]. However, no prior work has successfully applied GPGPU as a P4 target due to three major challenges. Firstly, a P4 application assumes “match+action” pipelines as its fundamental architecture, whereas a GPU lacks of dedicated “match+action” hardware. Secondly, as a coprocessor GPU

Figure 1: The Software Stack of P4GPU

carries limited memory space and runs at a lower clock frequency comparing with CPUs and NPUs. Thirdly, GPU excels at high parallelism computation such as table lookup and header matching however struggles in branching and dependency as in packet header parsing and state machine transitions.

To overcome these challenges, we propose a tool called P4GPU to map a P4 program to a heterogeneous CPU/GPU architecture. Our contributions in this work are three-fold: 1) to the best of our knowledge, this is the first work to map P4 applications onto a heterogeneous CPU/GPU target. We provide the P4GPU toolset to parse a P4 code and generate the desired GPU kernels, which could be a major step towards implementing a P4-to-GPU compiler; 2) we design a load-balanced heterogeneous framework to accelerate network functions in network data plane; 3) we explore different IP lookup and packet classification kernel designs on GPU for our heterogeneous architecture.

2. SOFTWARE DESIGN

Since table matching and packet classification have become the bottlenecks in packet forwarding nowadays [2, 4], we decide to map the hot spot of the P4 program, namely, the table “match+action” engine, to GPU for acceleration as shown in Figure 1. In particular, we take the P4 community provided “simple router” program as a use case to demonstrate our P4-to-GPU mapping process, and design a generic IP lookup engine and a packet classification engine on GPU that are used in the “match+action” step.

Mapping a P4 program to GPU involves three major steps as depicted in the block diagram in Figure 2: 1) P4 intermediate representation (IR) preparation - this step converts a complete P4 program into P4 IR which is stored in a clean and concrete OrderedDict Python data structure; 2) kernel configuration generation - once obtaining the IR of the P4 program, we apply a IR parser in the second step that reads the IR Python dictionary, finds the region of interest, and extract the target field parameters. We apply regular expression (RegEx) technique on IR to find the correct execution order of all tables; and 3) kernel initialization - in this step, we use the configuration parameters in step 2 to

Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ANCS '16 March 17-18, 2016, Santa Clara, CA, USA

© 2016 Copyright held by the owner/author(s).

ACM ISBN 978-1-4503-4183-7/16/03.

DOI: <http://dx.doi.org/10.1145/2881025.2889480>

Figure 2: Mapping P4 Program to GPU Kernels

Figure 3: Heterogeneous Packet Processing System Architecture

initialize the pre-defined GPU kernels.

We design two GPU kernel engines for the three tables (ipv4_lpm, forward, send_frame) in “simple router”: 1) lookup engine: we implement a generic Longest Prefix Match (LPM) kernel for IPv4 and IPv6 route lookup. Our baseline design is a linear search kernel, and we design two trie-based lookup kernels - binary trie and k-stride multibit trie for comparison; 2) classifier engine: we design a classifier engine with exact/wildcard-match capabilities. Our baseline design is linear search, and we also implement the grid-of-tries algorithm for optimized performance.

3. ARCHITECTURAL DESIGN

We depict the packet processing flow on the proposed architecture in Figure 3 and explain it in the following steps. Step 1: DMA directs the incoming packets that arrive at NIC to the system main memory; Step 2: CPU controls the incoming packets to fill packet buffers that will be fed into the GPU coprocessor by batches; step 3: depending on the behavior of incoming packet workload, load balancer (LB) on CPU will determine the percentage of workload that is offloaded to GPU and CPU. The design of LB is lightweight and responsive. Since GPU is more responsive to high packet rate, we implement a counter-based workload profiler to estimate the packet rate, and use the estimation to decide how many packets are offloaded to coprocessor; step 4: GPU and CPU process their share of workload and send buffered results back to main memory; step 5: CPU programs NIC to either forward or drop the processed packets.

4. PERFORMANCE EVALUATION

With commodity laptop hardware (CPU - Intel Quad Core i7-36100M; GPU - NVIDIA GT 650M with 384 cores), we use realistic publicly available datasets to evaluate our system performance. We apply the largest available prefix dataset CAIDE RouteView (January 2015) IPv4 with about 550,000 entries and IPv6 with about 20,000 entries for evaluating the GPU lookup engines. To test the performance of the classifier engine, we use ClassBench to generate a set of filters, such as FW (firewall) and ACL (Access Control List), in different sizes.

Figure 4: GPU Binary Trie Lookup Throughput Comparison with Different Batch Sizes

Figure 5: Performance of Different Classifier Kernel Design with Firewall Rules

We design two different network traffic generators for evaluation. The “ideal IO” generator assumes packets are already transmitted to the host memory, therefore no packet transmission overhead is involved. The “socket IO” generator assumes packets are transmitted from Click Modular Router to the host with a socket connection.

The performance metrics include throughput, average latency per packet, million lookups per second (MLPS), and million of classified packets per second (MCPS), etc. As demonstrated in Figure 4, the lookup engine throughput varies with various batch sizes. In the “ideal IO” case, system throughput can reach up to 580 Gbps for IPv4 and 390 Gbps for IPv6; and in the “socket IO” case, we observe that the maximum throughput is around 20 Gbps for both IPv4 and IPv6. With fixed batch size being 512, Figures 5 show the classifier kernel performance with linear search and grid-of-trie algorithm. We can observe that packet classification speed can reach as high as 93 MCPS with linear search algorithm for 500 firewall rules, and classification speed can still keep above 60 MCPS with grid-of-trie algorithm when the number of firewall rules is 4k.

5. REFERENCES

- [1] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.
- [2] P. Bosshart, G. Gibb, H. Kim, G. Varghese, N. McKeown, and M. Horowitz. Forwarding metamorphosis: Fast programmable match-action processing in hardware for sdn. In *ACM SIGCOMM 2013*. ACM, August 2013.
- [3] S. Han, K. Jang, K. Park, and S. Moon. Packetshader: A gpu-accelerated software router. *SIGCOMM Comput. Commun. Rev.*, 40(4):195–206, Aug. 2010.
- [4] G. Liao, H. Yu, and L. Bhuyan. A new ip lookup cache for high performance ip routers. In *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, pages 338–343, June 2010.