COP: Compiler Optimizations to Reduce Memory Stalls for Network Pipelines Written in P4

Shailja Pandey Ankit Bhardwaj Anmol Panda Sorav Bansal
{csz168117, ankitbhr.cstaff, ird11569, sbansal}@cse.iitd.ac.in
Indian Institute of Technology Delhi

1 Introduction

Software-based packet processing on general-purpose hardware is ubiquitous in virtualized data centers. Concurrently, novel high-level domain-specific languages (DSLs), for specifying modern packet processing pipeline functionality, have been proposed (e.g., P4 [2]). While these DSLs separate protocol-specification from switch implementation, and thus significantly improve productivity, they rely on an optimizing compiler to automatically convert the protocol-specification to a high performance switch implementation. We report our experiences with adding an optimizer to the P4C compiler [3] with a focus on reducing memory stalls. Our compiler translates a high-level P4 program to a lower-level C-based implementation that links with the DPDK infrastructure [1], and eventually gets executed on a multi-socket x86 machine.

Figure 1 shows the abstract forwarding model in P4 [2]. Typical packet-processing logic involves packet parsing (convert a packet to an efficiently accessible meta-data structure), and de-parsing (serialize the potentially modified meta-data into a packet), and a set of match-action tables. The match-action tables define rules to perform packet classification and determine the control flow. In a typical data-center networking, the number of rules in these tables is quite large making the look-up operations memory intensive in nature. The disparity in speeds of CPU and main memory causes stalls in the pipeline during look-up operations. Hence, optimizing the look-up operations reduces the per-packet CPU cycles and increases the throughput. In this work, we aim to ① reduce the number of look-up operations per packet and ② reduce the stall time per look-up operation. To reduce the number of look-up operations, we add an optimization pass in the compiler which selectively joins multiple match-action tables and performs lookup into the resulting table. Furthermore, scheduling and software prefetching help to reduce stall time per lookup by exploiting memory-level parallelism offered by the underlying architecture.

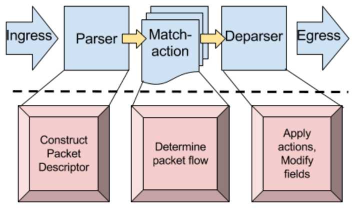

Figure 1: P4 forwarding model

2 Implementation

P4 programs can be represented in an accompanying high-level intermediate representation (HLIR), which facilitates easy analysis and transformation. Our compiler converts an HLIR program to C code and during this conversion, we apply our optimizations, implemented as two compiler passes.

TableCombine pass: This pass coalesces multiple tables connected in series into one table in the HLIR syntax. The joining of match-action tables is affected by dependencies between tables, type of lookup (exact, LPM, ternary, etc) and size of tables. The size of joined table is restricted by an upper limit for number of table entries, determined by the underlying data-structure. As part of this work, we join tables containing exact match look-ups and in future we plan to work on other types of lookups. The set of actions for the joined table is the union of all actions from the component tables. This optimization depends on the system's memory capacity and therefore, can't be applied for very large tables.

Scheduling and Software Prefetching: Most modern hardware support multiple in-flight memory requests simultaneously with the help of MSHRs(10 for our system). If the system exploits full MLP, the average memory access time is (one memory access time + δ)/10 i.e.

*Shailja Pandey is a student author

average memory access time is reduced by a factor of almost 10. To reduce the stall time even further, software prefetching can be used.

Instruction scheduler tries to issue independent instructions from reorder buffer. Generally, instructions from different packets are mutually independent. Therefore, processing multiple packets simultaneously increases the opportunity for the scheduler to issue independent instructions. Hence, COP transforms the code such that multiple packets pass through each stage together in the network pipeline. This compiler pass schedules the instructions, and adds prefetching instructions to ensure higher utilization of both CPU and memory. As we target lookup operations, COP focuses on scheduling memory related instructions to reduce average memory access time for each lookup.

3 Experiment Results

We use two Dell Poweredge R430 Rack Server, based on Haswell architecture. Each server has two dual port NICs, one Intel XL710(2x40 GbE) and one Intel x540(2x10 GbE), adding to a total capacity of 100GbE. We run applications (L2FWD, IPV4, IPV6, NDN, CRYPTO, and L2L3ACL) on one server and DPDK Pktgen traffic generator on the other. All the reported results are for 1 core and we expect linear scaling with multiple cores.

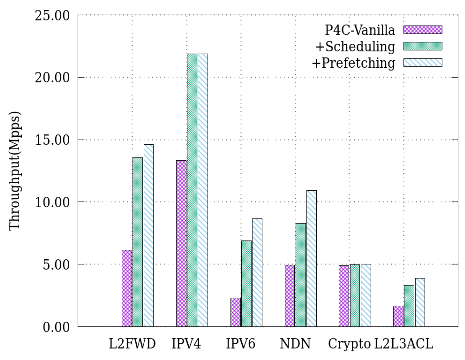

Figure 2: Effect of scheduling and prefetching

Effect of Scheduling and Prefetching: In figure 2, the "`+Scheduling`" results represent the case when COP schedules the instructions corresponding to a batch of packets to exploit memory level parallelism. The "`+Prefetching`" results represent the case when we have done scheduling as well as software prefetching to prefetch the data into cache memory. We obtain a total gain of 138% for L2FWD, 64% for IPv4, 280% for IPv6, 121% for NDN, 2% for CRYPTO and 135% for L2L3ACL application. As CRYPTO does not have any look-up operation, we do not see a gain whereas for all other applications, we obtain significant gain in performance.

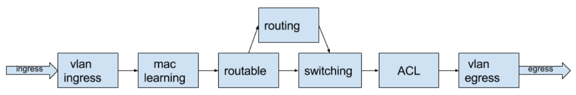

Figure 3: L2L3-ACL application control flow

Effect of TableCombine: As shown in figure 3, L2L3-ACL application[4] has seven table look-up operations with one probable diversion in the control flow. We performed the experiment with three different table sizes and we compare our results with individual tables generated by vanilla P4C[3].

#Entries	P4C	Scheduling+Prefetching	+Join
Min	2.9	4.8	7.4
Medium	2.7	4.8	6.9
Max	2.4	4.2	6.2

Table 1: Effect of optimizations on throughput(MPPS)

As table {4} contains a diversion and table {6} contains a ternary lookup, they cannot be joined with other tables. Consequently, there are only four table lookups after joining {1,2,3}, {4}, {5,7}, {6} tables as opposed to seven in base case. Table 1 shows resulting throughput after applying `TableCombine`, `Scheduling` and `Prefetching` passes. As the size of joined table is bigger than that of individual tables, scheduling optimization adds to the benefit obtained due to MLP.

References

[1] *Intel Data Plane Development Kit*. http://dpdk.org/.

[2] BOSSHART, P., DALY, D., GIBB, G., IZZARD, M., MCKEOWN, N., REXFORD, J., SCHLESINGER, C., TALAYCO, D., VAHDAT, A., VARGHESE, G., AND WALKER, D. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev. 44*, 3 (July 2014), 87–95.

[3] LAKI, S., HORPÁCSI, D., VÖRÖS, P., KITLEI, R., LESKÓ, D., AND TEJFEL, M. High speed packet forwarding compiled from protocol independent data plane specifications. In *Proceedings of the 2016 ACM SIGCOMM Conference* (New York, NY, USA, 2016), SIGCOMM '16, ACM, pp. 629–630.

[4] SHAHBAZ, M., CHOI, S., PFAFF, B., KIM, C., FEAMSTER, N., MCKEOWN, N., AND REXFORD, J. Pisces: A programmable, protocol-independent software switch. In *Proceedings of the 2016 ACM SIGCOMM Conference* (New York, NY, USA, 2016), SIGCOMM '16, ACM, pp. 525–538.