

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

Shailja Pandey, Ankit Bhardwaj, Anmol Panda, Sorav Bansal

Indian Institute of Technology Delhi, India



## Motivation

1. Novel high-level domain-specific languages (DSLs) for specifying modern packet processing pipeline functionality are deployed to separate the protocol-specification from underlying switch implementation. These DSLs rely on an optimizing compiler.
2. Manual optimization of such programs is undesirable as it requires highly skilled programmers, and is error-prone. This motivates us for the need to support optimization during compilation.
3. Prior work has focused on architecture-independent optimizations. We present a compiler that adds *architecture-specific optimizations* and compiles a high-level P4 program to a lower-level C-based implementation.

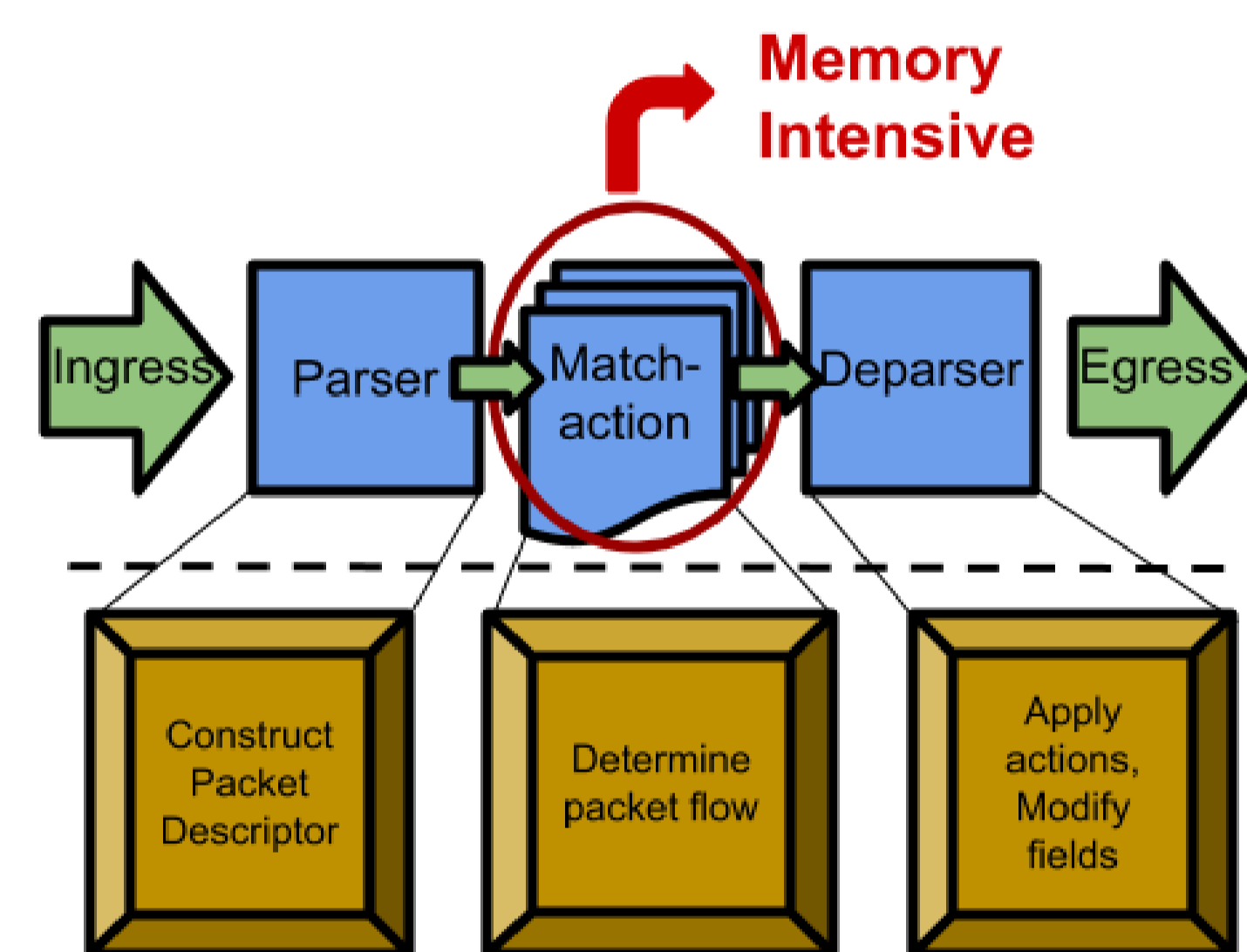


Figure 1: An abstract P4 forwarding model

## Objectives

We aim to optimize the memory-intensive **table look up operation** in the pipeline.

1. Reduce the number of lookup operations per packet by selectively joining look-up tables.
2. Study the impact of hardware characteristics like I/O device-memory interconnect, CPU-memory interconnect, CPU processing speed, and memory hierarchy on the performance of a given network processing program.
3. Reduce stall time per look-up operation by harnessing **memory level parallelism** through instruction scheduling and software prefetching.

## Compiler Optimizations

We deploy a two-phased procedure, a configuration phase and an optimization phase, to improve the application performance.

### Configuration Phase

1. In the configuration phase, the compiler measures the hardware characteristics of the system by running standard micro-benchmarks.
2. The characteristics measured include: DMA latency, degree of memory level parallelism, and L3 cache and memory latency for the underlying setup.

### Optimization Phase

The compiler deploys the optimizations in three passes.

#### 1. TableCombine Pass

- A. Identify type of lookup (exact, ternary, lpm, etc.)
- B. Divide the processing pipeline into equivalence classes based on inter-table dependencies
- C. Swap nodes to bring the 'E' nodes (exact lookup-type) together based on dependencies and feasibility of join
- D. Merge the nodes that can be joined

#### 2. DLIR Pass

- A. The choice of the data structure used for lookup-tables affects the efficiency of insert, delete and look-up operations
- B. The compiler determines the optimal data structure based on two factors:
  - a. Hardware characteristics learnt in the configuration phase
  - b. Table node specifications from the P4 HLIR

#### 3. Scheduling Pass

- A. Sophisticated instruction scheduling is required to exploit memory level parallelism
- B. The memory intensive operations of a look-up i.e. key extraction and hash lookup are executed for a batch of packets, thus allowing parallel memory accesses
- C. To further exploit MLP and remove memory stall, prefetching instructions are added

## Evaluation & Results

### Control-flow of L2L3-ACL application

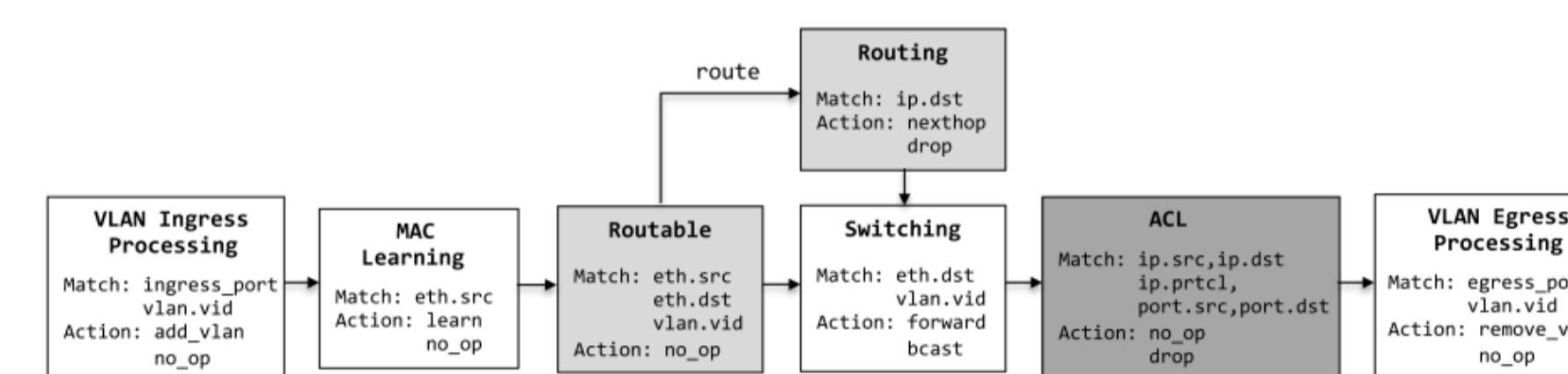


Figure 2: L2L3-ACL Control-flow[4]

### Performance gain due to optimizations

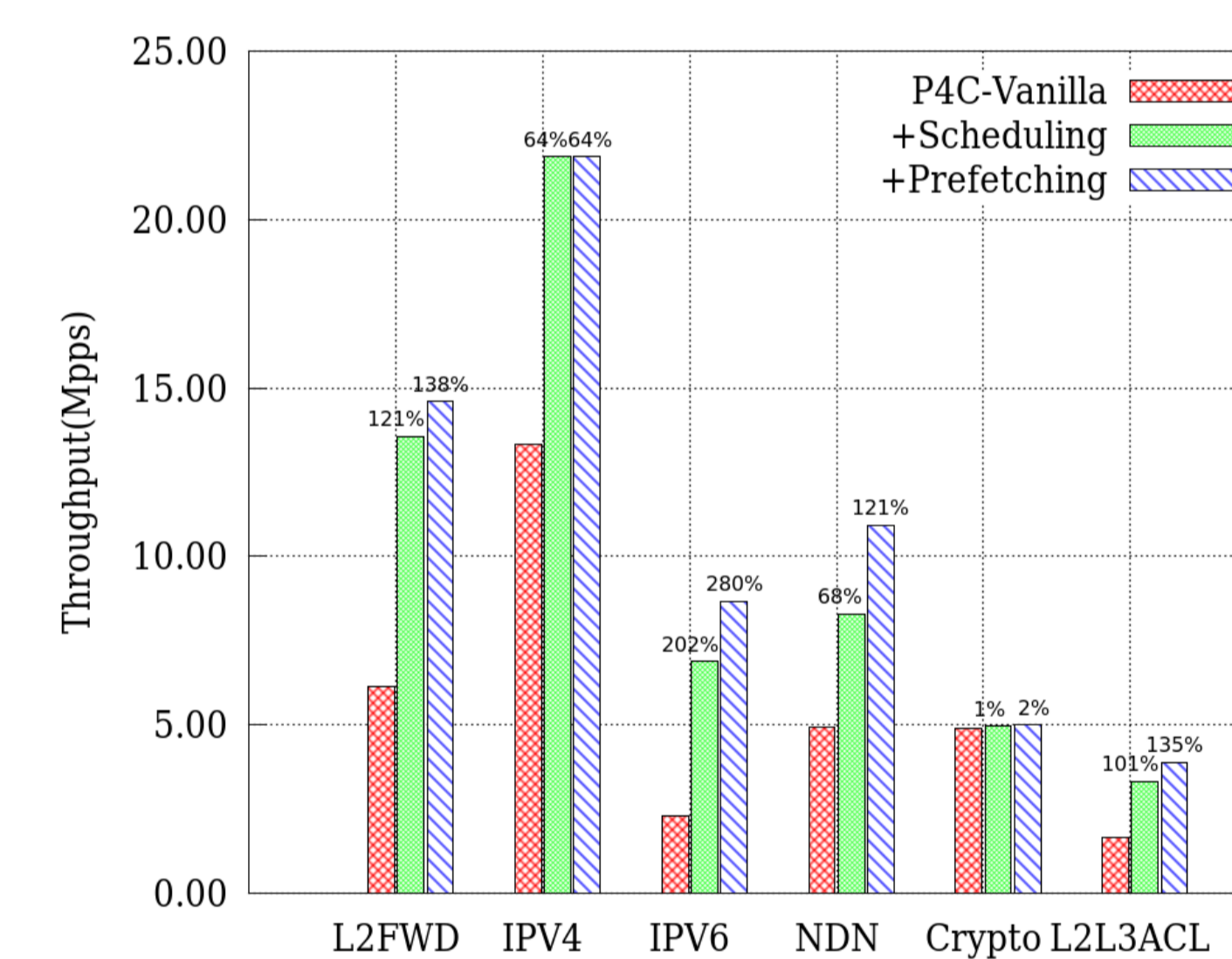


Figure 3: Effect of Prefetching and Batching optimizations

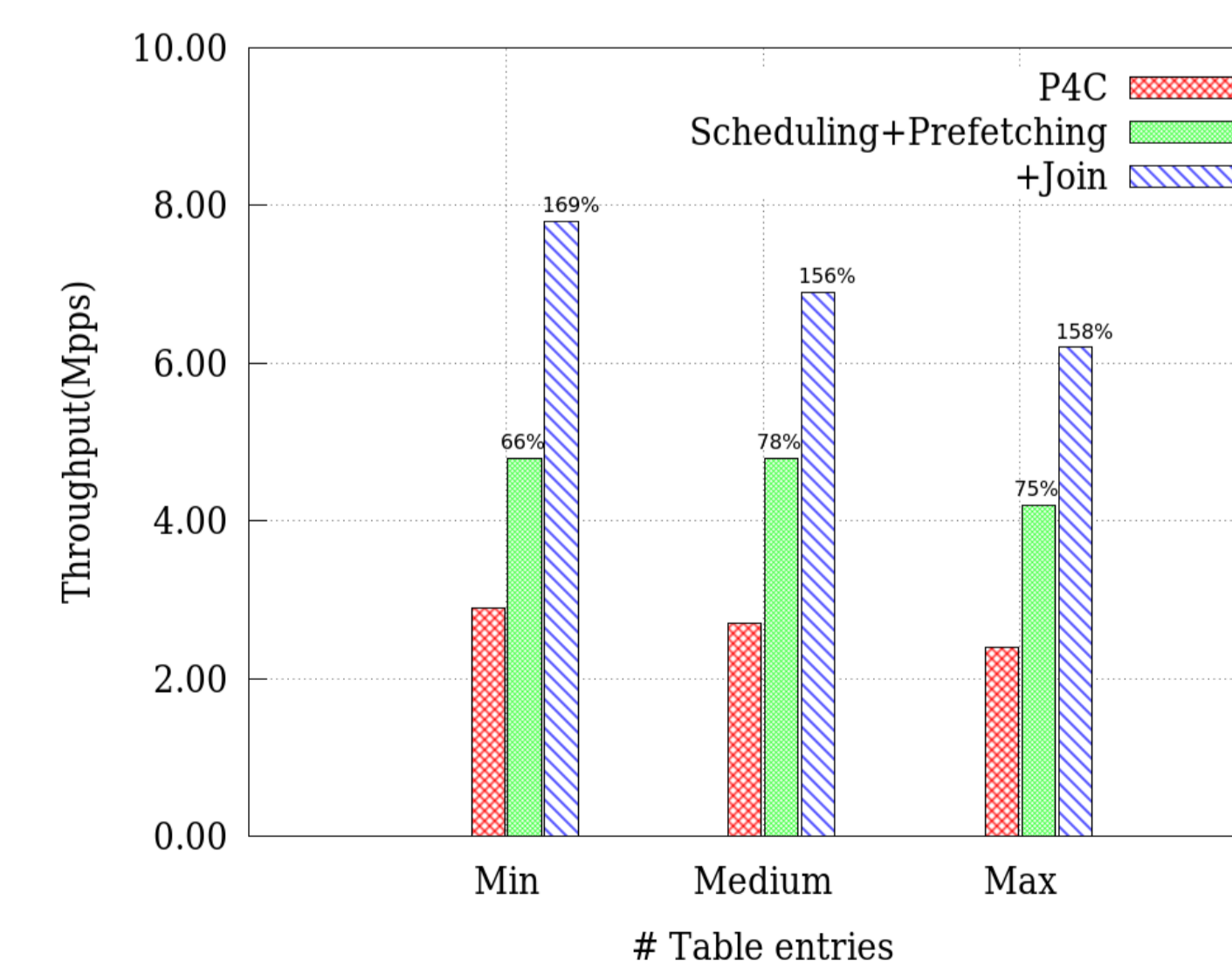


Figure 4: Effect of TableCombine optimization on L2-L3ACL application

## Conclusions

- Our results indicate that a compiler can generate architecture-specific optimized code for P4 applications when provided with information of available hardware resources and application characteristics.
- Our Scheduling and prefetching optimizations provide up to 280 percent gain for applications that were tested.
- For the L2L3-ACL application, the scheduling, prefetching and table-combine passes provide an overall gain of up to 169 percent.

## Forthcoming Research

We aim to extend this study\*:

- To measure the effect of architecture-dependent optimizations on fairly complex networking applications that involve multiple flow-diversions in their control-flow.
- To test the effectiveness of optimizations on different machine architectures.
- for applications which use stateful protocols.

## References

- [1] Intel Data Plane Development Kit. <http://dpdk.org/>.
- [2] Pat Bosshart, Dan Daly, Glen Gibb, Martin Izzard, Nick McKeown, Jennifer Rexford, Cole Schlesinger, Dan Talayco, Amin Vahdat, George Varghese, and David Walker. P4: Programming protocol-independent packet processors. *SIGCOMM Comput. Commun. Rev.*, 44(3):87–95, July 2014.
- [3] Sándor Laki, Dániel Horpácsi, Péter Vörös, Róbert Kitlei, Dániel Leskó, and Máté Tejfel. High speed packet forwarding compiled from protocol independent data plane specifications. In *Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16*, pages 629–630, New York, NY, USA, 2016. ACM.
- [4] Muhammad Shahbaz, Sean Choi, Ben Pfaff, Changhoon Kim, Nick Feamster, Nick McKeown, and Jennifer Rexford. Pisces: A programmable, protocol-independent software switch. In *Proceedings of the 2016 ACM SIGCOMM Conference, SIGCOMM '16*, pages 525–538, New York, NY, USA, 2016. ACM.

## Acknowledgements

Anmol Panda and Ankit Bhardwaj thank the Ministry of Human Resource Development (MHRD), Government of India for funding this research study.