P4GPU: A Study of Mapping a P4 Program onto GPU Target

Peilong Li, Tyler Alterio, Swaroop Thool and Yan Luo

ACANETS Lab (http://acanets.uml.edu/)

University of Massachusetts Lowell
Background & Motivation

• P4 enables programmability of network data plane

• Software-based routing and forwarding:
  • Programmability ↑
  • Line rate ↓

• P4 hardware targets:
  • CPU, NPU, reconfigurable switches, iNICs, FPGA, etc.
  • GPGPU potentiality? – Hundreds/thousands of cores, SIMD architecture, high memory bandwidth, mature parallel programming environment.
Contributions

• **P4-to-GPU**: provide a viable way to map a portion of the P4 code onto GPU target.

• **Acceleration**: design a heterogeneous framework with CPU and GPU to accelerate performance.

• **Optimization**: optimize lookup and classification kernels on GPU and explore load balancing on the CPU/GPU framework.
P4 to GPU: Overview

• Mapping P4 to GPU difficulties & hints

<table>
<thead>
<tr>
<th>Difficulties</th>
<th>Hints</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lack of dedicated hardware (match+action)</td>
<td>Take advantage of many cores</td>
</tr>
<tr>
<td>Inefficient kernel design leads to performance disaster</td>
<td>Kernel requires optimization; auto-generated kernel may not render good performance</td>
</tr>
<tr>
<td>Coprocessor has limited memory and low clock frequency</td>
<td>Heterogeneous CPU/GPU architecture is desired</td>
</tr>
<tr>
<td>GPU is good at high parallelism (table lookup, matching ...); bad at dependency and branching (parsing, state machine ...)</td>
<td>Only offload parallelizable code to GPU.</td>
</tr>
</tbody>
</table>
P4 to GPU: Software Stack

- Partially mapping
P4 to GPU: Design

• Three steps of mapping:
  
  Step 1: P4 intermediate representation preparation
  Step 2: GPU kernel configuration generation
  Step 3: GPU kernel initialization
The GPU kernel conf parser:
- Parse h.p4_tables etc. in IR to obtain P4 table conf
- Find the order of tables * from the control flow.

* Sequential tables for now
P4 to GPU: GPU Kernel Design

• LPM lookup kernel:
  • Used for “ipv4_lpm” table
  • Designed for both IPv4 and IPv6
  • Baseline kernel: linear search
  • Optimized kernel: binary trie and k-stride multibit trie

• Rule based classifier kernel:
  • Used for “forward” and “send_frame” table
  • Designed for both exact and wildcard match
  • Baseline kernel: linear search
  • Optimized kernel: grid-of-tries
P4 to GPU: GPU Kernel Design

• Latency hiding:
  • Batch processing: reduce number of data copying
  • 2D pipelining:

• Memory tweaking:
  • Use texture memory for storing lookup tables
  • Use mapped memory from host to device to reduce data movement overhead

• Data structure:
  • Hash tables/tries are not GPU-favored data structure
  • Trie-to-array vectorization
Architectural Design

• Components: CPU, main memory, GPU

Step 1: receiving
Step 2: batching
Step 3: load-balancing and offloading
Step 4: results buffering
Step 5: forwarding actions
Evaluation: Setup

• Experiment platform:
  • CPU: Intel Quad Core i7-3610QM @ 2.3 GHz
  • Main memory: 8 GB
  • GPU: Nvidia GT 650M with 384 cores

• Dataset:
  • RouteView (Jan 2015) IPv4 (550,000 entries), IPv6 (20,000 entries)
  • ClassBench: a set of filters, i.e. FW and ACL

• Traffic generation:
  • Random generator: assumes no packet IO overhead
  • Click Modular Router: emulated packet sending/receiving (socket-based)
Evaluation: Results

• Lookup Kernel

• Classifier Kernel
Evaluation: Results

• Throughput and Latency
Conclusion & Future Work

• P4GPU tool provides a viable way to map P4 code onto GPU target.
• GPU is a promising target for P4 to achieve high throughput and low latency forwarding.
• Integrate P4 table dependency to P4GPU tool
• Study the possibility of optimized GPU kernel auto generation.
• Implement GPU kernels for more functionalities, e.g. OpenFlow.
That’s All