ClickNP: Highly flexible and High-performance Network Processing with Reconfigurable Hardware

Bojie Li[§†] Kun Tan[†] Layong (Larry) Luo[‡] Yanqing Peng[•†] Renqian Luo[§†]
Ningyi Xu[†] Yongqiang Xiong[†] Peng Cheng[†] Enhong Chen[§]
[†]Microsoft Research [§]USTC [‡]Microsoft [•]SJTU
{v-bojli, kuntan, laluo, v-reluo, ningyixu, yqx, pengc}@microsoft.com, cheneh@ustc.edu.cn

1. INTRODUCTION

Highly flexible software network functions are critical components to enable multi-tenancy in cloud environments. However, software packet processing on a commodity server has limited capacity and induces high latency. While software network functions can be scaled out by using more servers, doing so adds significant cost. This demo targets to accelerate network functions with programmable hardware, *i.e.*, FPGA, which is now a mature technology and inexpensive for datacenter deployment.

Compared to CPU or GPU, FPGAs have a much lower clock frequency and a smaller memory bandwidth. However, unlike CPU or GPU, which has only limited cores, and thus limited parallelism, FPGAs have massive parallelism built-in. Modern FPGAs may have millions of logic gates, hundreds Kbit registers, tens of Mbits of BRAM, and thousands of DSP blocks, and in theory, each of them can work in parallel. Therefore, to achieve high performance, a programmer must fully utilize the massive parallelism inside FPGA.

Our goal is to build a versatile, high performance network function platform with FPGA-acceleration. First, the platform should be *fully programmed using high-level languages.* FPGA is predominately programmed using low-level hardware description languages (HDL), which is hard to program and difficult to debug, therefore almost not accessible for most software programmers. Second, we should support a *modular architecture* for packet processing. Third, network functions should handle packets flowing at the line-rates of 40/100Gbps with *ultra-low latency*. Finally, FPGA may not be suitable for all tasks and we should support fine-grained processing separation between CPU and FPGA, which requires *high-performance CPU-FPGA communication*.

This demo presents ClickNP, a FPGA-accelerated platform for highly flexible and high-performance network functions with commodity servers. ClickNP is highly flexible as it is completely programmable using high-level C-like languages, and exposes a familiar modular programming abstraction that resembles Click Modular Router [2]. ClickNP is also high-performance. Compared to existing software network functions, with FPGA, ClickNP improves the throughput by 10x, but also reduces the latency by 10x.

We have implemented the ClickNP tool-chain, which can integrate with various commercial HLS tools. We implemented and optimized about 100 common elements and used them to build three demo applications: (1) An Open vSwitch that supports L2 network virtualization with NVGRE. Additionally, the switch supports packet classification for firewall and strict priority queue for scheduling. (2) A stateful L4 load balancer that supports up to 32M concurrent flows and can accept 10M new flows per second. (3) As a final demo, we show that ClickNP is rather a general computing platform. We show a face detector in live video stream with convolutional neural network (CNN).

SIGCOMM '16, August 22-26, 2016, Florianopolis, Brazil

DOI: http://dx.doi.org/10.1145/2934872.2934897

2. SYSTEM ARCHITECTURE

ClickNP provides a modular architecture and the basic processing module is called an *element*. A ClickNP element has following properties: (1) Local *states* that are only accessible inside the element. (2) Input and output *ports*. (3) Handler functions: (a) an *initialization handler*, which is called only once when the element starts, (b) a *processing handler*, which is continuously called to check input ports and process the available data, and (c) a *signal handler*, which receives and processes the commands (*signals*) from the manager thread in the host program.

An output port of an element can connect to an input port of another element through a *channel*. In ClickNP, a channel is basically a FIFO buffer that is written to one end and read from the other end.

The ClickNP tool-chain contains a ClickNP compiler as front-end (*e.g.*, Visual Studio or GCC), and a C/C++ compiler and an HLS tool (*e.g.*, Altera OpenCL SDK or Xilinx Vivado HLS) as back-end. To write a ClickNP program, a developer needs to divide her code into three parts: (1) A set of *elements*, each of which implements a conceptually simple operation, (2) A *configuration file* that specifies the connectivity among these elements and whether an element runs on host or FPGA, and (3) A *host manager* that initializes each element and controls their behavior during the runtime, *e.g.*, according to the input of administrators. These three parts of source code are fed into the ClickNP compiler and translated into intermediate source files for both host program and FPGA program. Each channel between a host element and a FPGA element is compiled to a logical *slot* in PCIe DMA, which provides high-throughput (up to 25 Gbps) and low-latency (<2μs) streaming interface. *Signals* are sent through a special PCIe slot. The host program can be directly compiled by a normal C/C++ compiler, while the FPGA program is synthesized using commercial HLS tools.

ClickNP exploits all levels of parallelism in FPGA to speed up processing. First, packets flow from one element to another along a processing pipeline to exploit *task parallelism*. Bottleneck elements in the pipeline are duplicated to provide higher throughput by exploiting *data parallelism*. Second, as HLS tools schedule operations in an element into pipeline stages, we optimize elements to ensure each handler function can be fully pipelined, *i.e.*, process a datum in every clock cycle. To achieve this maximal throughput,

we (1) use registers whenever possible because registers have no latency, while memory access has one cycle latency; (2) Leverage *delayed write* and *memory scattering* techniques to remove read-write memory dependency; (3) Completely *unroll* loops to trade area for speed; (4) Balance pipeline stages by *offloading* slow path (*e.g.* DRAM access) to another element.

3. DEMONSTRATION

3.1 Open vSwitch

Network virtualization is critical to multi-tenant data centers. Software virtual switches on CPU, *e.g.* Open vSwitch [4] cannot catch up with 40Gbps line-rate for small packets. This demo showcases a ClickNP implementation of major functions in Open vSwitch. Each lookup table and action is an element in ClickNP. Parsed packet header fields, Openflow tags and intermediate states during table lookup are carried in metadata channels among elements. Packet data is carried in flit channel beside metadata channel. Each packet is processed independently in pipeline, and there is no caching for "known flows".

The FPGA processes packets *bump-in-the-wire* between host NIC and the top-of-rack (ToR) switch. One Ethernet port of the FPGA is connected to the ToR switch, the other Ethernet port of the FPGA is connected to the host NIC. VMs are configured with SR-IOV NIC interfaces. Packets for different VMs are identified by VLAN tags. Packets to another VM within same host is loop-backed with VLAN tag changed. Packets to ToR switch are encapsulated with NVGRE header according to address mapping in tenant network. Packets from ToR switch are decapsulated, applied VLAN tag and forwarded to NIC.

Besides conventional OVS functions, to show the flexibility of ClickNP, we implement a strict priority scheduler as an Openflow action to enable pFabric [1] scheduling. In demo scenario, two sender VMs on a same physical host transfer data to a receiver VM on another physical host in the same rack. One sender VM sends short flows, while the other sends long flows. A TCP option encodes total flow size in each packet. If no scheduling is performed in Open vSwitch, the egress queue to ToR is a drop-tail FIFO. When pFabric scheduling is turned on for ToR egress queue, packets from short flows will have higher priority and less likely to be dropped. As a result, the flow completion time (FCT) of short flows will improve greatly, while the 40G link is still fully utilized.

The demo OVS has four Openflow tables in a pipeline for packet classification. As in Openflow, tags can be pushed and popped within the pipeline, which allows efficient representation of IP and port sets, *e.g.* drop packets from intranet IP set to prohibited port set. First two Openflow tables are HashTCAM with 16K entries and at most 16 distinct bit-masks. HashTCAM divides the table space into 16 1K hash tables, each of which associated with a bit-mask. Other two tables are BRAM-based true TCAM with 1K entries. It partitions 104-bit key into 13 8-bit words. Each word corresponds to 2^8 bit vectors indicating match result of each rule against every possible word. The Openflow tables can update 300K rules per second via *signals* without impacting data plane throughput.

3.2 Stateful L4 Load Balancer

The Open vSwitch is stateless. In this demo, we show that a stateful network function can also be easily implemented. We pick L4 load balancer as an example, as it requires all packets from a same flow be forwarded to a same backend server (DIP), and the load balancing policy may be more complex than ECMP. This demo showcases stateful packet processing with a Layer-4 load balancer corresponding to MUX in Ananta [3]. A *state manager* looks up flow 5-tuple to determine flow state. For the first packet of each flow, an element *DIPAlloc* allocates a backend server according to load balancing policy. If a second packet comes before allocation is done, the packet is dropped to avoid duplicate allocation. For remaining packets of the flow, backend server index is extracted from flow state. Flow state is invalidated when it receives a FIN packet, or timeout occurs. FPGA encapsulates an IP-in-IP header to direct the packet to assigned backend server. We run the *DIPAlloc* element on CPU because the allocation policy is potentially complex. All other elements run on FPGA for high throughput and low latency.

The state manager needs 32 bytes per flow to hold 5-tuple, timestamp and backend server index. Flow entries are saved in a hash table with 64M slots, consuming 2GB of DRAM. The hash table uses linear lookup to resolve conflicts. To speedup flow lookup, we maintain a 4-way associative flow cache in BRAM with 16K cache lines, replaced using Least-Recently Used (LRU) algorithm.

In this demo, synthetic TCP flows are generated by ClickNP TrafficGen in another FPGA. The host program of TrafficGen controls offered packet rate, throughput, number of concurrent flows and flow size distribution. The host program of L4 load balancer shows processed packet rate, throughput, average latency, total number of flows and flow arrival rate.

We also demonstrate debugging an element on host CPU. We first annotate the element in question with "host" in ClickNP configuration file, then add some debug prints in the element, re-compile, and the element in question will run on host CPU and print to console.

3.3 Face Detection and Alignment with CNN

In addition to network processing, ClickNP is a general framework for joint CPU-FPGA processing. This demo showcases face detection and alignment [5] implemented with ClickNP. RGB image streams from web camera to host elements that perform face detection using haar-like features, then streams to FPGA elements that perform face alignment using a 9-layer convolutional neural network (CNN). Finally detection result is streamed back to host. Each convolution layer and FC layer is implemented as an element in ClickNP. Parameters are stored in DRAM with tiling to leverage sequential read for throughput. An image *double buffer* in BRAM before each layer allows parallel communication and computation.

Video from Web camera is streamed to the CPU via USB cable. Host elements run on CPU and stream intermediate results to FPGA elements via PCIe I/O channnel. Classification results are transferred back to the host program via PCIe I/O channel.

4. REFERENCES

[1] M. Alizadeh et al. pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 Conference*, SIGCOMM '13, pages 435–446, New York, NY, USA, 2013. ACM.

[2] E. Kohler et al. The click modular router. *ACM Transactions on Computer Systems (TOCS)*, 18(3):263–297, 2000.

[3] P. Patel et al. Ananta: Cloud scale load balancing. In *Proceedings of the ACM SIGCOMM 2013 Conference*, SIGCOMM '13, pages 207–218, New York, NY, USA, 2013. ACM.

[4] B. Pfaff et al. The design and implementation of open vswitch. In *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pages 117–130, 2015.

[5] J. Qiu et al. Going deeper with embedded fpga platform for convolutional neural network. In *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '16, pages 26–35, New York, NY, USA, 2016. ACM.

5. DEMO REQUIREMENTS

FPGA needs to run in a server which is hard to be shipped to the conference. So we will connect to our demo servers in Beijing via remote desktop from our laptops. We will prepare videos for all demos in case of network failure.

For the face detection demo, we may bring a credit-card size SoC board and a Web camera which are connected to USB ports of our laptop and do not require separate power supply.

In addition to the default demo settings (table space and a poster board), we hope to have:

- Wired Internet connection for stable remote desktop connection.
- Two monitors to show the audience with larger screen. (Better to have HDMI interface so that we do not buy HDMI to VGA converters.)

Space needed: Table space for two laptops and two monitors.

Demo setup time: We wish to test network connectivity at the scene before the day of demo. If network connectivity is good, we can setup the demo in 30 minutes.

Thanks for your support!