ClickNP: Highly Flexible and High Performance Network Processing with Reconfigurable Hardware

Bojie Li\textsuperscript{1,2}, Kun Tan\textsuperscript{1}, Layong (Larry) Luo\textsuperscript{1}, Yanqing Peng\textsuperscript{1,3}, Renqian Luo\textsuperscript{1,2}, Ningyi Xu\textsuperscript{1}, Yongqiang Xiong\textsuperscript{1}, Peng Cheng\textsuperscript{1}, Enhong Chen\textsuperscript{2}

\textsuperscript{1}Microsoft Research, \textsuperscript{2}USTC, \textsuperscript{3}SJTU
Virtualized network functions

Dedicated hardware NFs are not flexible

Virtualized NFs on servers to maximize flexibility
Scale-up challenges for software NF

Limited processing capacity
Number of CPU cores needed for 40 Gbps line rate

<table>
<thead>
<tr>
<th>Network function</th>
<th>Implementation</th>
<th>1500B pkt @ 40 Gbps (normal case)</th>
<th>64B pkt @ 40 Gbps (worst-case estimate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVGRE tunnel encapsulation</td>
<td>Hyper-V virtual switch</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>Firewall (8K rules)</td>
<td>Linux iptables</td>
<td>21</td>
<td>480</td>
</tr>
</tbody>
</table>
Scale-up challenges for software NF

Limited processing capacity
Number of CPU cores needed for 40 Gbps line rate

<table>
<thead>
<tr>
<th>Network function</th>
<th>Implementation</th>
<th>1500B pkt @ 40 Gbps (normal case)</th>
<th>64B pkt @ 40 Gbps (worst-case estimate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVGRE tunnel encapsulation</td>
<td>Hyper-V virtual switch</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>Firewall (8K rules)</td>
<td>Linux iptables</td>
<td>21</td>
<td>480</td>
</tr>
</tbody>
</table>

Inflated and unstable latency
Add tens of microseconds to milliseconds latency to data plane
Latency may grow to milliseconds under high load
1 ms occasional delay would violate SLA (e.g., trading services)
FPGA in the cloud

FPGA-based SmartNIC

Bump-in-the-wire processing between NIC and ToR switch\textsuperscript{[1]}

\textsuperscript{[1]} SIGCOMM’15 keynote (also the image source)
FPGA in the cloud

FPGA-based SmartNIC

Bump-in-the-wire processing between NIC and ToR switch[1]

Why FPGA?

Massive parallelism
- Millions of logic elements
- Thousands of memory blocks

Low power consumption (~20W)

General computing platform (vs. GPU, NP) to accelerate various cloud services

Mature technology with a reasonable price

[1] SIGCOMM’15 keynote (also the image source)
FPGA challenge: *Programmability*

Hardware description language (HDL): push many software developers away

```verilog
always @(posedge SYSCLK or negedge RST_B)
begin
  if(!RST_B)
    DMA_TX_DATA <= `UD 8'hff;
  else
    DMA_TX_DATA <= `UD DMA_TX_DATA_N;
end

// send: "hello world !"
always @(posedge SYSCLK)
begin
  if (rst) begin
    dma_data <= 88'h0;
    dma_valid <= 1'b0;
  end
end
```

Ahhhhhhhhhhhhhhhh!
Project ClickNP

Making FPGA accessible to software developers

- **Flexibility**: fully programmable using high-level language

- **Modularized**: Click abstractions familiar to software developers; easy code reuse

- **High performance**: high throughput; microsecond-scale latency

- **Joint CPU/FPGA packet processing**: FPGA is no panacea; fine-grained processing separation
Programming model

as if programming on a multi-core processor

cores (elements) running in parallel

communicate via channels, not shared memory
Element: single-threaded core

- **states**
  - (reg/mem)

- **Process handler**
  - (main thread)

- **Signal handler**
  - (ISR)

- Inputs:
  - input channels (I/O)

- Outputs:
  - output channels (I/O)

- Signal/Control from host:
  - (interrupt)
Extension 1: CPU element

Write once, run on both FPGA and CPU
Compile to a logic block on FPGA or a binary running on CPU
Enable joint CPU/FPGA programming
Simplify debugging

PCle I/O channel
Enable fast communication between elements on FPGA and CPU
Low latency: 1 µs
High throughput: 25.6 Gbps (PCle Gen2 x8)
Extension 2: Verilog Element

Embedding native Verilog code in ClickNP element

Enable hybrid C/Verilog programming
(analog to C/ASM in CPU world)
**Architecture**

Cross-platform toolchain
- Altera OpenCL / Vivado HLS
- Visual Studio / GCC
Architecture

Cross-platform toolchain
Altera OpenCL / Vivado HLS
Visual Studio / GCC
Architecture

ClickNP host process
- Mgr thrd
- Worker thrd

ClickNP library
- PCIe I/O channel
- vendor libs

ClickNP compiler

ClickNP host mgr

ClickNP shell
- ClickNP
  - vendor specific runtime

FPGA

Host

Catapult PCIe Driver

Vendor HLS

Cross-platform toolchain
- Altera OpenCL / Vivado HLS
- Visual Studio / GCC

Intermediate C files

C compiler

Verilog

Vendor libs

ClickNP elements

ClickNP script
Architecture

ClickNP host process
  Mgr thrd  Worker thrd

ClickNP library
  PCIe I/O channel  vendor libs

Host
  Catapult PCIe Driver

ClickNP host process

ClickNP host mgr

ClickNP library

ClickNP compiler

C compiler

vendor HLS

Intermediate C files

Cross-platform toolchain
  Altera OpenCL / Vivado HLS
  Visual Studio / GCC

Catapult shell
  ClickNP

vendor specific runtime

Cross-platform toolchain
  Altera OpenCL / Vivado HLS
  Visual Studio / GCC
Architecture

Cross-platform toolchain
Altera OpenCL / Vivado HLS
Visual Studio / GCC
Architecture

ClickNP host process
  Mgr thrd  Worker thrd

ClickNP library
  PCIe I/O channel  vendor libs

ClickNP compiler

ClickNP host mgr

ClickNP script

C compiler

vendor HLS

vendor specific runtime

Cross-platform toolchain
Altera OpenCL / Vivado HLS
Visual Studio / GCC
Example: Packet logger

ClickNP Configuration:

1. `Count :: cnt @`
2. `Tee :: tee`
3. `host PktLogger :: logger`
4. `from_tor -> cnt -> tee [1] -> to_tor`
5. `tee [2] -> logger`
Example: Packet logger

ClickNP Configuration:

```plaintext
count cnt @
tee tee
host PktLogger : logger
from_tor -> cnt -> tee [1] -> to_tor
tee [2] -> logger
```

Count element:

```plaintext
element Count <1, 1> {
    state{
        ulong count;
    }
    init{
        count = 0;
    }
    handler{
        if (get_input_port() != PORT_1) {
            return (PORT_1);
        }
        flit x;
        x = read_input_port(PORT_1);
        if (x.fd.sop) count = count + 1;
        set_output_port(PORT_1, x);
        return (PORT_1);
    }
    signal{
        CI Signal p;
        p.Sig.LParam[0] = count;
        set_signal(p);
    }
}
```
Parallelizing in FPGA

Parallelism across elements
Pipeline parallelism
Data parallelism

Parallelism inside an element
Minimize memory dependency
  Use registers
  Delayed write
  Memory scattering
Balance pipeline stages
  Unroll loops
  Offload slow path to another element
Parallelism across elements

Pipeline parallelism

Duplicate pipeline to leverage data parallelism
Parallelism inside element (1)

Element is synthesized into a pipeline
Parallelism inside element (2)

Principle 1: Minimize memory dependency

Memory dependency:
Read cannot start before last write, because read and write addresses may conflict.
Parallelism inside element (3)

*Delayed write* to remove read-write dependency

Memory read and write can operate in parallel:
Read in.addr, Write buf.addr
Different memory addresses!

Delayed write:
Buffer new data in a register
Delay memory write until next read
Parallelism inside element (4)

Principle 2: Balance pipeline stages

DRAM read is an unbalanced pipeline stage. Pipeline is scheduled with a fixed interval to cover the worst-case latency. So throughput is bottlenecked by DRAM read latency.
Parallelism inside element (5)

Offload slow path to another element

Original

Fast path

Slow path

Loop

Read Cache

Hit?

Read DRAM

Output

Y

N

Loop

Read Cache

Hit?

Y

N

To slow path

From slow path

Output

To fast path

From fast path

Read DRAM
Parallelism inside element (6)

Offload slow path to another element

3 cache hits:

1 cache hit, 1 cache miss, 1 cache hit:
## ClickNP element library

<table>
<thead>
<tr>
<th>Element</th>
<th>Fmax (MHz)</th>
<th>Peak Throughput</th>
<th>Delay (cycles)</th>
<th>Resource LE %</th>
<th>Resource BRAM %</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4_Parser</td>
<td>221.9</td>
<td>113.6 Gbps</td>
<td>11</td>
<td>0.8%</td>
<td>0.2%</td>
</tr>
<tr>
<td>IPChecksum</td>
<td>226.8</td>
<td>116.1 Gbps</td>
<td>18</td>
<td>2.3%</td>
<td>1.3%</td>
</tr>
<tr>
<td>NVGRE_Encap</td>
<td>221.8</td>
<td>113.6 Gbps</td>
<td>9</td>
<td>1.5%</td>
<td>0.6%</td>
</tr>
<tr>
<td>AES_CTR</td>
<td>217.0</td>
<td>27.8 Gbps</td>
<td>70</td>
<td>4.0%</td>
<td>23.1%</td>
</tr>
<tr>
<td>SHA1</td>
<td>220.8</td>
<td>113.0 Gbps</td>
<td>105</td>
<td>7.9%</td>
<td>6.6%</td>
</tr>
<tr>
<td>CuckooHash</td>
<td>209.7</td>
<td>209.7 Mpps</td>
<td>38</td>
<td>2.0%</td>
<td>65.5%</td>
</tr>
<tr>
<td>HashTCAM</td>
<td>207.4</td>
<td>207.4 Mpps</td>
<td>48</td>
<td>18.7%</td>
<td>22.0%</td>
</tr>
<tr>
<td>LPM_Tree</td>
<td>221.8</td>
<td>221.8 Mpps</td>
<td>181</td>
<td>4.3%</td>
<td>13.2%</td>
</tr>
<tr>
<td>SRPrioQueue</td>
<td>214.5</td>
<td>214.5 Mpps</td>
<td>41</td>
<td>2.6%</td>
<td>0.6%</td>
</tr>
<tr>
<td>RateLimiter</td>
<td>141.5</td>
<td>141.5 Mpps</td>
<td>14</td>
<td>16.9%</td>
<td>14.1%</td>
</tr>
</tbody>
</table>

Nearly 100 elements
20% re-factored from Click modular router
Cover packet parsing, checksum, tunnel encap/decap, crypto, hash tables, prefix matching, packet scheduling, rate limiting...

Throughput: 200 Mpps / 100 Gbps
Mean delay: 0.19 us, max delay: 0.8 us
Mean LoC: 80, max LoC: 196
ClickNP element library

<table>
<thead>
<tr>
<th>Element</th>
<th>Fmax (MHz)</th>
<th>Peak Throughput</th>
<th>Delay (cycles)</th>
<th>Resource LE %</th>
<th>Resource BRAM %</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4_Parser</td>
<td>221.9</td>
<td>113.6 Gbps</td>
<td>11</td>
<td>0.8%</td>
<td>0.2%</td>
</tr>
<tr>
<td>IPChecksum</td>
<td>226.8</td>
<td>116.1 Gbps</td>
<td>18</td>
<td>2.3%</td>
<td>1.3%</td>
</tr>
<tr>
<td>NVGRE_Encap</td>
<td>221.8</td>
<td>113.6 Gbps</td>
<td>9</td>
<td>1.5%</td>
<td>0.6%</td>
</tr>
<tr>
<td>AES_CTR</td>
<td>217.0</td>
<td>27.8 Gbps</td>
<td>70</td>
<td>4.0%</td>
<td>23.1%</td>
</tr>
<tr>
<td>SHA1</td>
<td>220.8</td>
<td>113.0 Gbps</td>
<td>105</td>
<td>7.9%</td>
<td>6.6%</td>
</tr>
<tr>
<td>CuckooHash</td>
<td>209.7</td>
<td>209.7 Mpps</td>
<td>38</td>
<td>2.0%</td>
<td>65.5%</td>
</tr>
<tr>
<td>HashTCAM</td>
<td>207.4</td>
<td>207.4 Mpps</td>
<td>48</td>
<td>18.7%</td>
<td>22.0%</td>
</tr>
<tr>
<td>LPM_Tree</td>
<td>221.8</td>
<td>221.8 Mpps</td>
<td>181</td>
<td>4.3%</td>
<td>13.2%</td>
</tr>
<tr>
<td>SRPrioQueue</td>
<td>214.5</td>
<td>214.5 Mpps</td>
<td>41</td>
<td>2.6%</td>
<td>0.6%</td>
</tr>
<tr>
<td>RateLimiter</td>
<td>141.5</td>
<td>141.5 Mpps</td>
<td>14</td>
<td>16.9%</td>
<td>14.1%</td>
</tr>
</tbody>
</table>

Nearly 100 elements
20% re-factored from Click modular router
Cover packet parsing, checksum, encap/decap, hash tables, prefix matching, rate limiting, crypto, packet scheduling...

Throughput: 200 Mpps / 100 Gbps
Mean delay: 0.19 us, max delay: 0.8 us
Mean LoC: 80, max LoC: 196
## ClickNP element library

<table>
<thead>
<tr>
<th>Element</th>
<th>Fmax (MHz)</th>
<th>Peak Throughput</th>
<th>Delay (cycles)</th>
<th>Resource LE %</th>
<th>Resource BRAM %</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4_Parser</td>
<td>221.9</td>
<td>113.6 Gbps</td>
<td>11</td>
<td>0.8%</td>
<td>0.2%</td>
</tr>
<tr>
<td>IPChecksum</td>
<td>226.8</td>
<td>116.1 Gbps</td>
<td>18</td>
<td>2.3%</td>
<td>1.3%</td>
</tr>
<tr>
<td>NVGRE_Encap</td>
<td>221.8</td>
<td>113.6 Gbps</td>
<td>9</td>
<td>1.5%</td>
<td>0.6%</td>
</tr>
<tr>
<td>AES_CTR</td>
<td>217.0</td>
<td>27.8 Gbps</td>
<td>70</td>
<td>4.0%</td>
<td>23.1%</td>
</tr>
<tr>
<td>SHA1</td>
<td>220.8</td>
<td>113.0 Gbps</td>
<td>105</td>
<td>7.9%</td>
<td>6.6%</td>
</tr>
<tr>
<td>CuckooHash</td>
<td>209.7</td>
<td>209.7 Mpps</td>
<td>38</td>
<td>2.0%</td>
<td>65.5%</td>
</tr>
<tr>
<td>HashTCAM</td>
<td>207.4</td>
<td>207.4 Mpps</td>
<td>48</td>
<td>18.7%</td>
<td>22.0%</td>
</tr>
<tr>
<td>LPM_Tree</td>
<td>221.8</td>
<td>221.8 Mpps</td>
<td>181</td>
<td>4.3%</td>
<td>13.2%</td>
</tr>
<tr>
<td>SRPrioQueue</td>
<td>214.5</td>
<td>214.5 Mpps</td>
<td>41</td>
<td>2.6%</td>
<td>0.6%</td>
</tr>
<tr>
<td>RateLimiter</td>
<td>141.5</td>
<td>141.5 Mpps</td>
<td>14</td>
<td>16.9%</td>
<td>14.1%</td>
</tr>
</tbody>
</table>

Nearly 100 elements
20% re-factored from Click modular router
Cover packet parsing, checksum, encap/decap, hash tables, prefix matching, rate limiting, crypto, packet scheduling...

Throughput: 200 Mpps / 100 Gbps
Mean delay: 0.19 us, max delay: 0.8 us
Mean LoC: 80, max LoC: 196
## ClickNP element library

<table>
<thead>
<tr>
<th>Element</th>
<th>Fmax (MHz)</th>
<th>Peak Throughput</th>
<th>Delay (cycles)</th>
<th>Resource LE %</th>
<th>Resource BRAM %</th>
</tr>
</thead>
<tbody>
<tr>
<td>L4_Parser</td>
<td>221.9</td>
<td>113.6 Gbps</td>
<td>11</td>
<td>0.8%</td>
<td>0.2%</td>
</tr>
<tr>
<td>IPChecksum</td>
<td>226.8</td>
<td>116.1 Gbps</td>
<td>18</td>
<td>2.3%</td>
<td>1.3%</td>
</tr>
<tr>
<td>NVGRE_Encap</td>
<td>221.8</td>
<td>113.6 Gbps</td>
<td>9</td>
<td>1.5%</td>
<td>0.6%</td>
</tr>
<tr>
<td>AES_CTR</td>
<td>217.0</td>
<td>27.8 Gbps</td>
<td>70</td>
<td>4.0%</td>
<td>23.1%</td>
</tr>
<tr>
<td>SHA1</td>
<td>220.8</td>
<td>113.0 Gbps</td>
<td>105</td>
<td>7.9%</td>
<td>6.6%</td>
</tr>
<tr>
<td>CuckooHash</td>
<td>209.7</td>
<td>209.7 Mpps</td>
<td>38</td>
<td>2.0%</td>
<td>65.5%</td>
</tr>
<tr>
<td>HashTCAM</td>
<td>207.4</td>
<td>207.4 Mpps</td>
<td>48</td>
<td>18.7%</td>
<td>22.0%</td>
</tr>
<tr>
<td>LPM_Tree</td>
<td>221.8</td>
<td>221.8 Mpps</td>
<td>181</td>
<td>4.3%</td>
<td>13.2%</td>
</tr>
<tr>
<td>SRPrioQueue</td>
<td>214.5</td>
<td>214.5 Mpps</td>
<td>41</td>
<td>2.6%</td>
<td>0.6%</td>
</tr>
<tr>
<td>RateLimiter</td>
<td>141.5</td>
<td>141.5 Mpps</td>
<td>14</td>
<td>16.9%</td>
<td>14.1%</td>
</tr>
</tbody>
</table>

Nearly 100 elements
20% re-factored from Click modular router
Cover packet parsing, checksum, encap/decap, hash tables, prefix matching, rate limiting, crypto, packet scheduling...

Throughput: 200 Mpps / 100 Gbps
Mean delay: 0.19 us, max delay: 0.8 us
Mean LoC: 80, max LoC: 196
Sample network functions

Each NF takes about one week to develop

<table>
<thead>
<tr>
<th>Network Function</th>
<th>Lines of Code *</th>
<th>Number of Elements</th>
<th>Resource LE %</th>
<th>Resource BRAM %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pkt generator</td>
<td>13</td>
<td>6</td>
<td>16%</td>
<td>12%</td>
</tr>
<tr>
<td>Pkt capture</td>
<td>12</td>
<td>11</td>
<td>8%</td>
<td>5%</td>
</tr>
<tr>
<td>OpenFlow firewall</td>
<td>23</td>
<td>7</td>
<td>32%</td>
<td>54%</td>
</tr>
<tr>
<td>IPSec gateway</td>
<td>37</td>
<td>10</td>
<td>35%</td>
<td>74%</td>
</tr>
<tr>
<td>L4 load balancer</td>
<td>42</td>
<td>13</td>
<td>36%</td>
<td>38%</td>
</tr>
<tr>
<td>pFabric scheduler</td>
<td>23</td>
<td>7</td>
<td>11%</td>
<td>15%</td>
</tr>
</tbody>
</table>

* ClickNP configuration file
Case study: IPSec gateway (1)

Why IPSec datapath offloading?

CPU is the bottleneck for computation intensive processing
Even with AES-NI, software AES processing is far from 40 Gbps
Currently X86 has no SHA-1 acceleration instructions
Case study: IPSec gateway (1)

Why IPSec datapath offloading?
CPU is the bottleneck for computation intensive processing
Even with AES-NI, software AES processing is far from 40 Gbps
Currently X86 has no SHA-1 acceleration instructions

Challenges to FPGA offloading
AES
• Single AES element throughput is 27.8 Gbps (< 40 Gbps)
SHA-1
• Packet payload is split into 64-byte data blocks
• Dependency between successive data blocks in a packet
• Only 1.07 Gbps if processed sequentially!
Case study: IPSec gateway (2)

Solution: Parallelizing in FPGA

AES-256-CTR: Leverage data parallelism
Case study: IPSec gateway (2)

Solution: Parallelizing in FPGA
AES-256-CTR: Leverage data parallelism

SHA-1: Leverage packet-level parallelism
Process data blocks from different packets in parallel
Case study: IPSec gateway (2)

Solution: Parallelizing in FPGA

AES-256-CTR: Leverage data parallelism

SHA-1: Leverage packet-level parallelism

Process data blocks from different packets in parallel
Case study: IPSec gateway (3)

Single-tunnel performance

<table>
<thead>
<tr>
<th></th>
<th>ClickNP</th>
<th>StrongSwan / Linux (out of the box)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>37.8 Gbps</td>
<td>628 Mbps</td>
</tr>
<tr>
<td>Latency</td>
<td>13 us (stable)</td>
<td>50us ~ 5ms</td>
</tr>
</tbody>
</table>
Case study: L4 load balancer (1)

Requires per-flow states for many flows

32M concurrent flows in off-chip DRAM
16K flow cache in on-chip BRAM
Case study: L4 load balancer (1)

Requires per-flow states for many flows
32M concurrent flows in off-chip DRAM
16K flow cache in on-chip BRAM

Requires complex nexthop allocation policy
Nexthop allocation in CPU element: joint CPU/FPGA processing
Case study: L4 load balancer (2)

Performance
10M new flows per second
51Mpps for 8K flows
11Mpps for 32M flows
4µs stable latency
Area cost

Overhead compared to hand-written Verilog

<table>
<thead>
<tr>
<th>NetFPFA Function</th>
<th>Resource Utilization</th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>LUTs</td>
<td>Registers</td>
<td>BRAMs</td>
</tr>
<tr>
<td>Input arbiter</td>
<td></td>
<td>2.1x / 3.4x</td>
<td>1.8x / 2.8x</td>
<td>0.9x / 1.3x</td>
</tr>
<tr>
<td>Output queue</td>
<td></td>
<td>1.4x / 2.0x</td>
<td>2.0x / 3.2x</td>
<td>0.9x / 1.2x</td>
</tr>
<tr>
<td>Header parser</td>
<td></td>
<td>0.9x / 3.2x</td>
<td>2.1x / 3.2x</td>
<td>N/A</td>
</tr>
<tr>
<td>Openflow table</td>
<td></td>
<td>0.9x / 1.6x</td>
<td>1.6x / 2.3x</td>
<td>1.1x / 1.2x</td>
</tr>
<tr>
<td>IP checksum</td>
<td></td>
<td>4.3x / 12.1x</td>
<td>9.7x / 32.5x</td>
<td>N/A</td>
</tr>
<tr>
<td>Encap</td>
<td></td>
<td>0.9x / 5.2x</td>
<td>1.1x / 10.3x</td>
<td>N/A</td>
</tr>
</tbody>
</table>
## Area cost

### Overhead compared to hand-written Verilog

<table>
<thead>
<tr>
<th>NetFPFA Function</th>
<th>Resource Utilization Min / Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUTs</td>
</tr>
<tr>
<td>Input arbiter</td>
<td>2.1x / 3.4x</td>
</tr>
<tr>
<td>Output queue</td>
<td>1.4x / 2.0x</td>
</tr>
<tr>
<td>Header parser</td>
<td>0.9x / 3.2x</td>
</tr>
<tr>
<td>Openflow table</td>
<td>0.9x / 1.6x</td>
</tr>
<tr>
<td>IP checksum</td>
<td>4.3x / 12.1x</td>
</tr>
<tr>
<td>Encap</td>
<td>0.9x / 5.2x</td>
</tr>
</tbody>
</table>
## Area cost

### Overhead compared to hand-written Verilog

<table>
<thead>
<tr>
<th>NetFPFA Function</th>
<th>Resource Utilization Min / Max</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>LUTs</td>
</tr>
<tr>
<td>Input arbiter</td>
<td>2.1x / 3.4x</td>
</tr>
<tr>
<td>Output queue</td>
<td>1.4x / 2.0x</td>
</tr>
<tr>
<td>Header parser</td>
<td>0.9x / 3.2x</td>
</tr>
<tr>
<td>Openflow table</td>
<td>0.9x / 1.6x</td>
</tr>
<tr>
<td>IP checksum</td>
<td>4.3x / 12.1x</td>
</tr>
<tr>
<td>Encap</td>
<td>0.9x / 5.2x</td>
</tr>
</tbody>
</table>

### Area cost will be less of a concern

Quickly growing FPGA capacity: 1x (2013), 2.5x (2015), 10x (2018)
Conclusion

FPGA in the cloud
Fast, general and mature

ClickNP: *programming hardware as software*

*Highly flexible:* Program with high-level language
*Modular:* Click-like abstractions
*High performance:* Utilize massive parallelism in FPGA
*Joint CPU/FPGA packet processing*

Start from network processing, but it goes beyond...

General framework for programming on FPGA
Azure storage, Bing search, machine learning...
Thank you!
Questions!
Check out our demo on Thursday!
### Acceleration in host

**Offloading software NFs to accelerator**

<table>
<thead>
<tr>
<th></th>
<th>GPU</th>
<th>NP</th>
<th>FPGA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput</td>
<td>High</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td>Latency</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>Power</td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td>General computing</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
Performance of PCIe I/O channel
High level synthesis not sufficient

HLS tools (e.g. Vivado HLS)
- Only auxiliary tools for HDL tool chains
- Generated hardware modules (IP cores) need to be integrated manually

OpenCL (e.g. Altera)
- The OpenCL programming model is designed for batch processing in GPU => high latency
- Packet processing should use stream processing model
- Does not support joint CPU/FPGA processing well

Click2NetFPGA
- Low performance because not fully pipelined
- Unable to update configuration while data plane running
cross-platform toolchain

three parts of source files
Define elements ➔ ClickNP extended C
Define a configuration of elements ➔ ClickNP script language
Host manager program ➔ C/C++

make tool to glue all compile procedures
Source files (.cl/.cfg/.cpp) ➔ ClickNP compiler (clc) ➔ intermediate source files (.cl/.cpp)
FPGA program ➔ aoc/Vivado
Host program ➔ Microsoft cl + OpenCL libs

platform supported
Windows/Linux, Altera/Xilinx
Why do not simply use OpenCL?

OpenCL is for batch processing on GPU

Communicate by sharing memory

*Shared memory is the bottleneck!*

*Batch processing has large latency!*
Programming model

ClickNP is for stream processing on FPGA

Channels: more efficient than shared memory

Do not communicate by sharing memory; instead, share memory by communicating.

-- The slogan of Go language
Parallelism inside element (2)

Memory scattering: avoid false dependency
Parallelism inside element (3)

Unroll loops to balance pipeline stages
Parallelism inside element (4)

Offload slow path to another element

Throughput constrained by slow path

After offloading: