Design patterns for code reuse in HLS packet processing pipelines

Haggai Eran[*†], Lior Zeno[*], Zsolt István[‡], and Mark Silberstein[*]

[*]Technion — Israel Institute of Technology
[†]Mellanox Technologies
[‡]IMDEA Software Institute

FCCM 2019

Network packet processing & FPGAs

- High-throughput
- Low latency
- Predictable performance
- Flexibility

E.g.

- AccelNet on Microsoft Azure [Firestone et al. NSDI'18]

Network packet processing on FPGAs is hard!

- Require hardware design expertise
- Lack of software-like reusable libraries

Compared to CPU:

Click Modular Router

*There are three great virtues of a programmer: **Laziness**, Impatience and Hubris.*

Larry Wall

Creator of the Perl programming language

Why focus on high-level synthesis (HLS)?

- Abstract underlying hardware details
  - Automatic scheduling & pipelining
  - Reuse a design on different hardware
- High level language features (objects & polymorphism)

Focus on Xilinx Vivado HLS (C++)

How is HLS used for packet processing?

- Data-flow design
  - A fixed graph of independent elements
  - Operate on data when inputs are ready
  - Examples: [Blott ’13], [XAPP1209 ‘14], [Sidler ’15], ClickNP [Li ’16].

Our methodology focuses on data-flow designs.

Why is it hard to build an HLS networking lib?

- Only a subset of C++ is synthesizable.
  - Virtual functions cannot be used.
- Strict interfaces and patterns for performance.

Our `ntl` library overcomes these problems.

`ntl`: Networking Template Library

- New methodology for developing reusable data-flow HLS elements.
- Template class library that applies our methodology for network packet processing applications.

How to build reusable data-flow element pattern?

- Basic elements
  - C++ classes for each data-flow element
  - State kept as member variables
  - `step()` method implements functionality
  - Inline methods embedded in the caller
  - All interfaces are `hls::stream` (members/parameters)
- Reuse with customization via functional programming
- Composed through aggregation: reusable sub-graph.

Networking Template Library (`ntl`)

Class library of packet processing building blocks.

Category	Classes
Header processing elements	`pop/push_header, push_suffix`
Data-structures	`array, hash_table`
Scheduler	`scheduler`
Basic elements	`map, scan, fold, dup, zip, link`
Specialized stream wrappers	`pack_stream, pfifo, stream<Tag>`
Control-plane	`gateway`

Networking Template Library (`ntl`)

Class library of packet processing building blocks.

Category	Classes
Header processing elements	`pop/push_header, push_suffix`
Data-structures	`array, hash_table`
Scheduler	`scheduler`
Basic elements	`map, scan, fold, dup, zip, link`
Specialized stream wrappers	`pack_stream, pfifo, stream<Tag>`
Control-plane	`gateway`

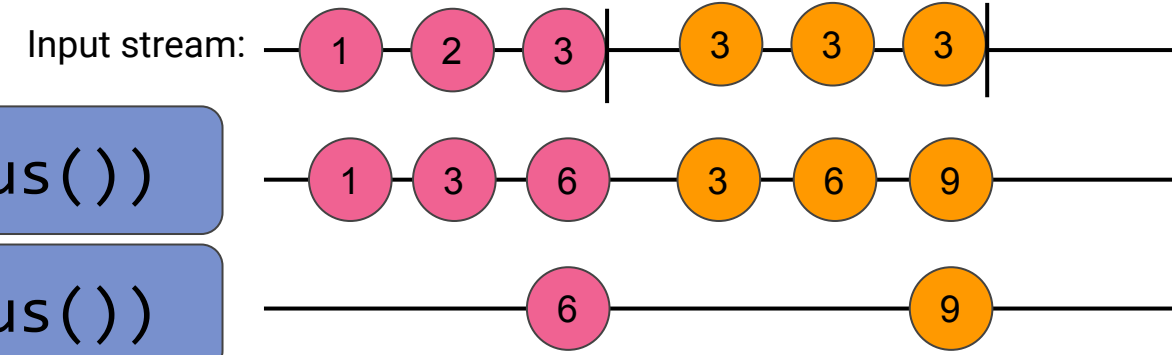
Example: scan and fold

Common operators in functional and reactive programming.

Modified to reset state for every packet.

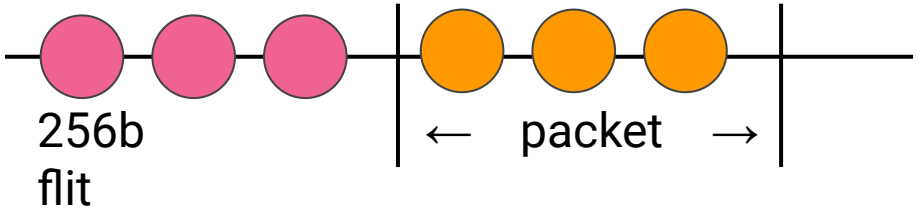
```
scan.step(input, plus())
```

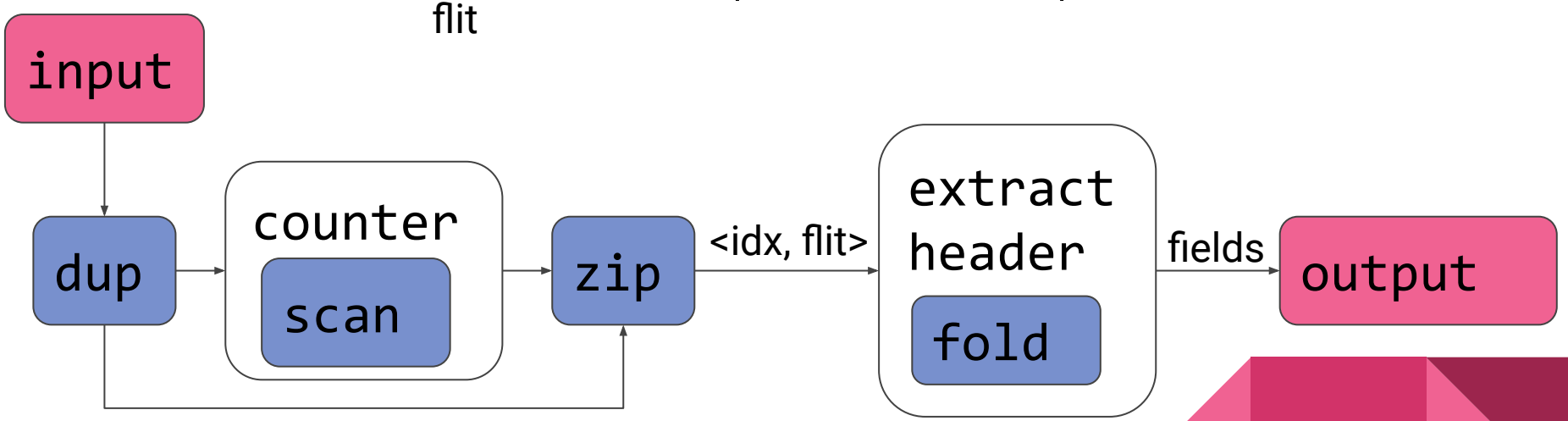
```
fold.step(input, plus())
```



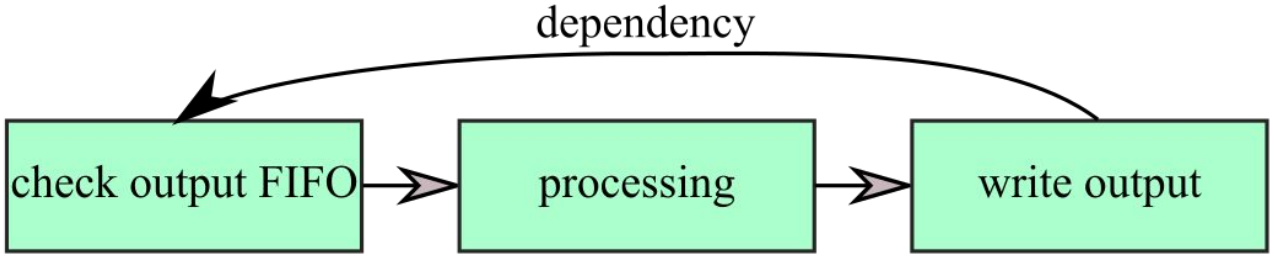
Can serve basis for more complex operators.

Fold & scan usage: parser example





Programmable threshold FIFO



Dependency between FIFO check and write → decreased throughput

`hls::stream` replacement

Evaluation

- How does `ntl` compare against legacy HLS, P4?
- Can we build a relatively complex application with `ntl`?

Targeting Mellanox Innova Flex SmartNIC

- Xilinx Kintex UltraScale XCKU060 FPGA
- Shell dictates 216.25 MHz clock rate
- Mellanox ConnectX-4 Lx ASIC NIC

Stateless UDP firewall example

Use hash-table to classify packets.

	Thpt.	Latency	LUTs	FFs	BRAM	LoC
HLS/`ntl`	72 Mpps	25 cycles	5296	7179	12	218
HLS legacy	72 Mpps	16 cycles	4087	4287	12	593
P4 (SDNet 2018.2)	108 Mpps	211 cycles	34531	49042	193	92

Stateless UDP firewall example

Use hash-table to classify packets.

All exceed line rate (59.5 Mpps)

	Thpt.	Latency	LUTs	FFs	BRAM	LoC
HLS/`ntl`	72 Mpps	25 cycles	5296	7179	12	218
HLS legacy	72 Mpps	16 cycles	4087	4287	12	593
P4 (SDNet 2018.2)	108 Mpps	211 cycles	34531	49042	193	92

Stateless UDP firewall example

Use hash-table to classify packets.

x2.7 less lines of code compared to legacy

	Thpt.	Latency	LUTs	FFs	BRAM	LoC
HLS/`ntl`	72 Mpps	25 cycles	5296	7179	12	218
HLS legacy	72 Mpps	16 cycles	4087	4287	12	593
P4 (SDNet 2018.2)	108 Mpps	211 cycles	34531	49042	193	92

Stateless UDP firewall example

Use hash-table to classify packets.

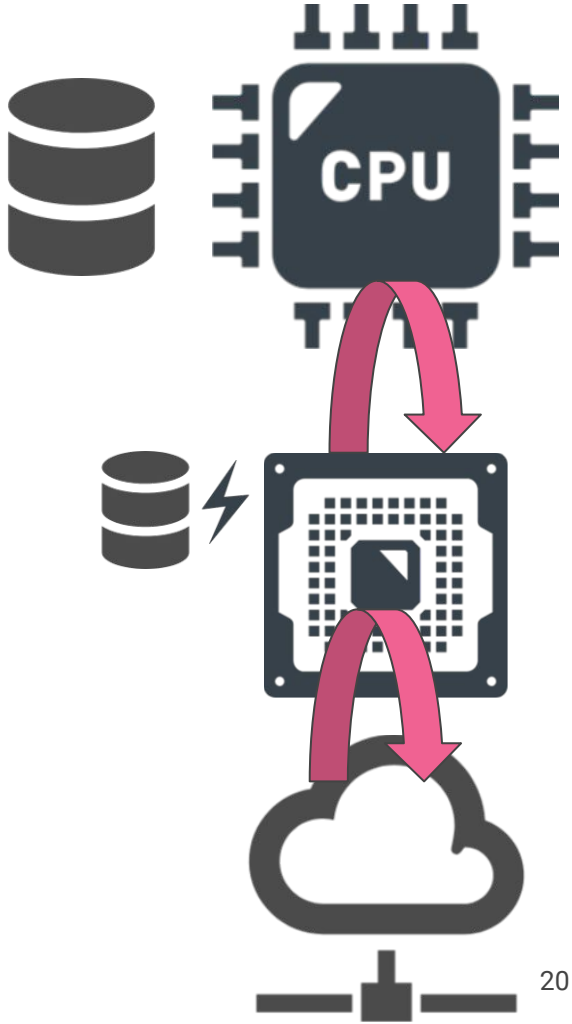
	Thpt.	Latency	LUTs	FFs	BRAM	LoC
HLS/`ntl`	72 Mpps	25 cycles	5296	7179	12	218
HLS legacy	72 Mpps	16 cycles	4087	4287	12	593
P4 (SDNet 2018.2)	108 Mpps	211 cycles	34531	49042	193	92

`ntl` requires more LoC, but improves latency & area

Key-value store cache

- Cache memcached values on SmartNIC.
- GET hits served directly from cache.
- Multi-tenant support.

Uses the NICA framework [ATC'19]
Both NICA and KVS cache use `ntl`.

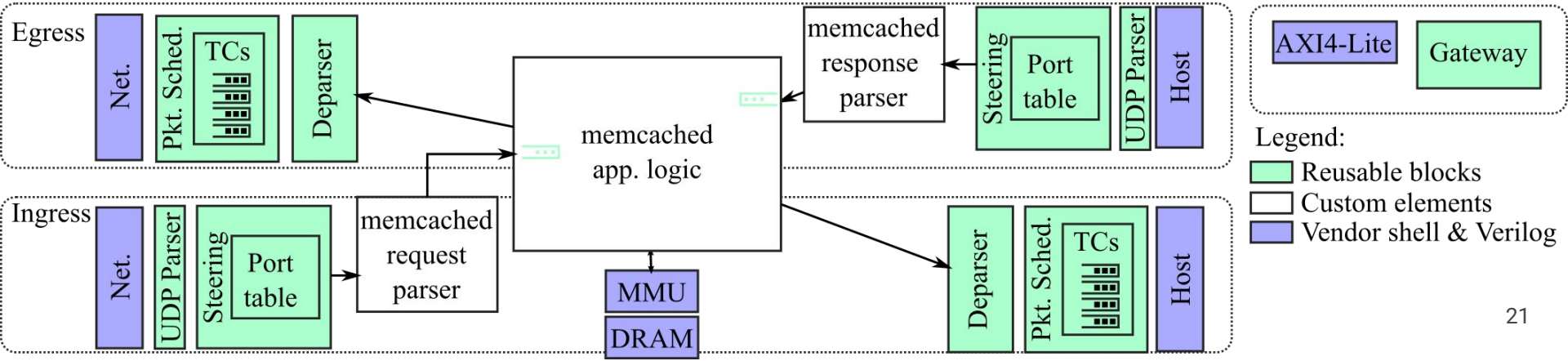


Key-value store cache

Processes 16-byte GET hits at 40.3 Mtps.

For 75% hit rate: 9× compared to CPU-only.

Uses: hash tables, header processing, scheduler, control plane, programmable FIFOs, ...



Related work: HLS methodology

- Xilinx application note [XAPP1209 ‘14]
  We adapt a similar data-flow design, but improve code reuse.
- Improving high-level synthesis with decoupled data structure optimization, [Zhao ’16].
  We similarly wrap data-structures, but remain within C++.
- Module-per-Object: a human-driven methodology for C++-based high-level synthesis design, [Silva ’19].
  Complementary methodology; we share some aspects but focus on data-flow packet processing and provide `ntl`.

Related work

Packet processing DSLs / libraries: P4 [Wang ’17], [Silva ’18], [SDNet], ClickNP [Li ’16], Emu [Sultana ’17], Maxeler.
We focus on general purpose C++ for its flexibility.

Dataflow HLS designs: Image/video processing [Oezkan ‘17], [OpenCV], HPC designs [de Fine Licht ‘18].
Higher order functions in HLS: [Thomas ‘16], [Richmond ‘18].
We apply similar techniques to packet processing.

Conclusion

We show a methodology for reusable packet processing in HLS, and create reusable building blocks for line-rate processing in the `ntl` library.

Try out `ntl`: https://github.com/acsl-technion/ntl

Thank you!

Questions?