




Accelerating Regular-Expression Matching on FPGAs with High-Level Synthesis



Mission-Critical Computing
NSF CENTER FOR SPACE, HIGH-PERFORMANCE,
AND RESILIENT COMPUTING (SHREC)

IWOCL 21

Devon Callanan

Luke Kljucaric

Alan George

University of Pittsburgh

NSF SHREC



Outline

Goals, Motivations, Challenges

Background

Kernel Approach

Bandwidth Scaling

Results

Conclusions, Future Work

Goals, Motivations, and Challenges

- **Goals**
 - Accelerate packet inspection for high-speed networking
 - Investigate performance characteristics of OpenCL
- **Motivations**
 - Faster link speeds demand faster deep packet inspection
 - FPGA design tools continue to improve
- **Challenges**
 - Throughput scaling costs grow exponentially
 - Tradeoffs between complexity and performance of high-level synthesis designs

Outline

Goals, Motivations, Challenges

Background

Kernel Approach

Bandwidth Scaling

Results

Conclusions, Future Work

What is a regex?

- Regex describe complex patterns in compact ways
- Queries are built using **metacharacters**, **grouping symbols**, and **regular characters**
- Example:
 - *colou?r*: Matches both American “color” and British “colour”
- Used in network intrusion detection, genomics, and natural language processing

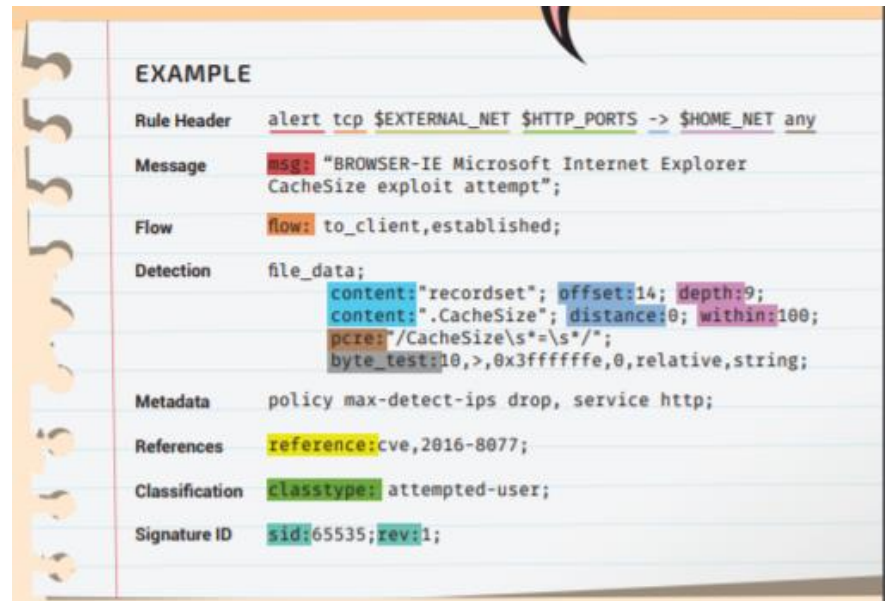
MetaChar	Meaning
	Boolean Or
*	Zero or More
+	One or More
?	Zero or One



shutterstock.com • 770544518

What is network intrusion detection?

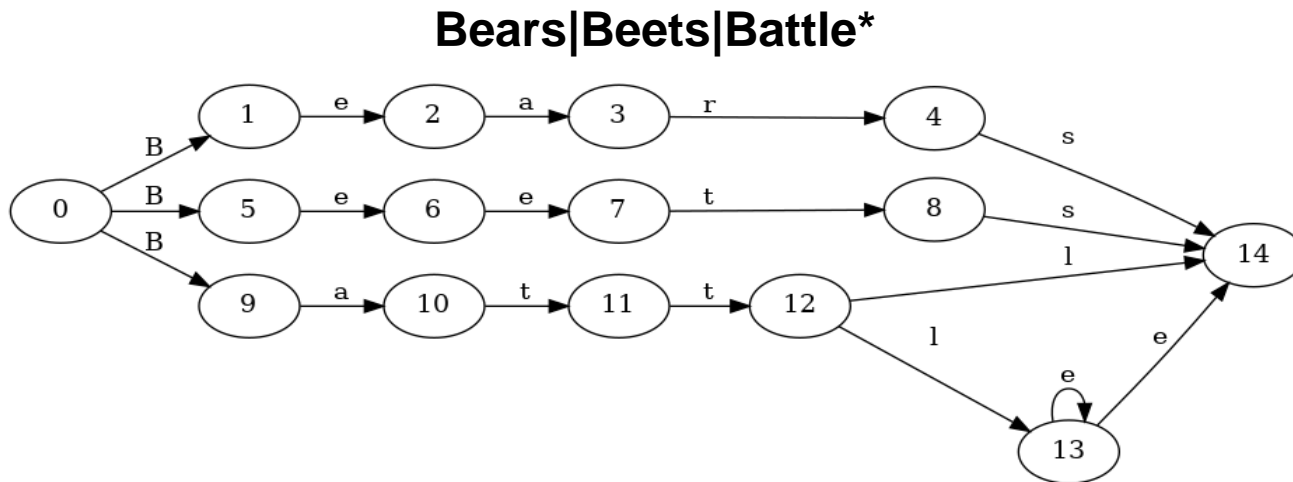
- Network intrusion detection systems (NIDS) watch network traffic for suspicious packets
- Security professionals write rules defining which packets should set off alerts in the system
- Can completely block some traffic
- Each rule can specify a regex as detection options
- SNORT is a common open-source NIDS with thousands of rules



```
EXAMPLE
Rule Header  alert tcp $EXTERNAL_NET $HTTP_PORTS -> $HOME_NET any
Message      msg: "BROWSER-IE Microsoft Internet Explorer
              CacheSize exploit attempt";
Flow         flow: to_client,established;
Detection    file_data;
              content:"recordset"; offset:14; depth:9;
              content:".CacheSize"; distance:0; within:100;
              pcre: /CacheSize\s*=\s*/;
              byte_test:10,>,0x3fffffff,0,relative,string;
Metadata     policy max-detect-ips drop, service http;
References   reference:cve,2016-8077;
Classification classtype: attempted-user;
Signature ID  sid:65535; rev:1;
```

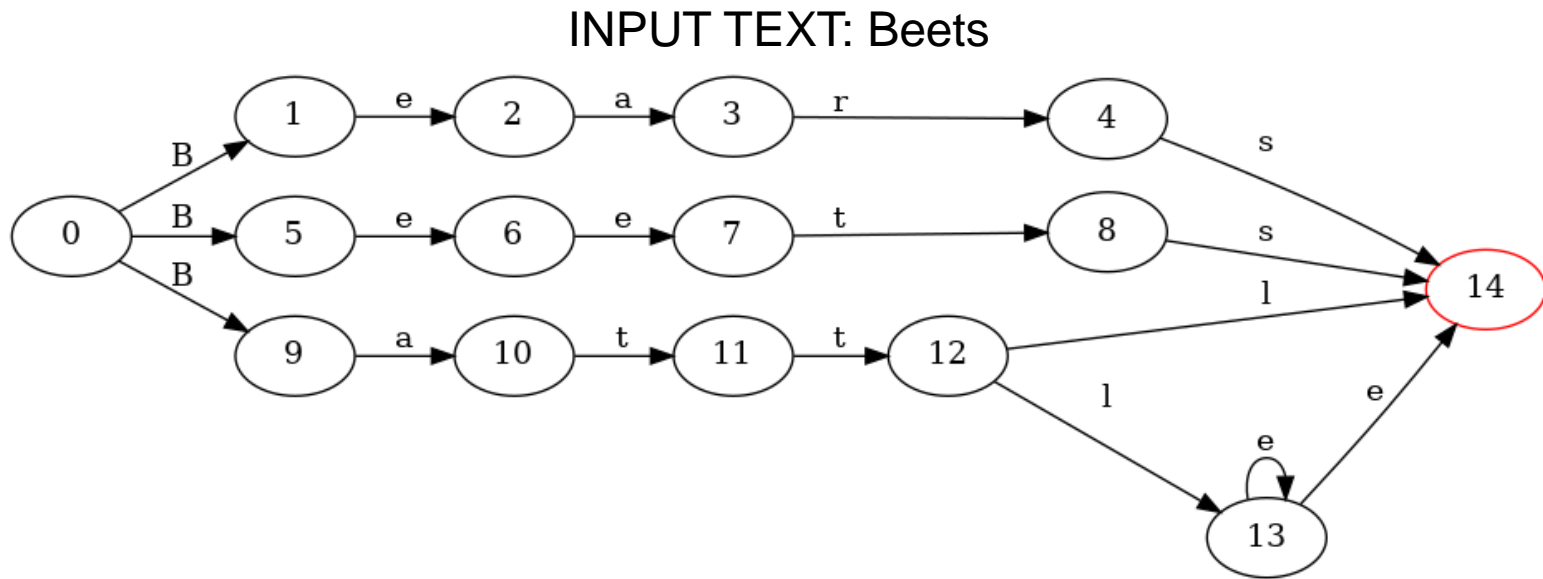
What is an NFA?

- Non-deterministic finite automata (NFAs) are state machines
 - Each circle is a state
 - Each arrow is a transition between states
- NFAs can be constructed to reach an **accepting state** only after certain input patterns
- The input patterns (regular language) described by an NFA can be described by an equivalent regex!



NFA Example

- For each input character, check if an active state can make a transition
- Update states in preparation for next input



What is high-level synthesis?

- Traditional FPGA development describes hardware constructs
 - Known as Register Transfer Level (RTL)
 - VHDL and Verilog common examples
- High-level synthesis (HLS) abstracts some fine-grained optimizations to ease development and allow rapid prototyping
- Intel provides access to HLS through OpenCL and oneAPI
 - Specifications developed by Khronos Group
 - oneAPI based on SYCL



Putting it all together...

Accelerating

- We want to handle enterprise-level data at enterprise-level speeds

Regular-Expression Matching

- We are using regex from the popular NIDS SNORT
- We are transforming these regex into state machines (NFAs)

on FPGAs

- We are accelerating the state machine computations on FPGAs

with High-Level Synthesis

- We are creating the accelerators with high-level synthesis

Outline

Goals, Motivations, Challenges

Background

Kernel Approach

Bandwidth Scaling

Results

Conclusions, Future Work

HLS Kernels - Structure

- HLS allows for C-like coding of accelerators
- Representing state machines requires states and transitions
 - States are Boolean local variables (single bit per state)
 - Transitions are combinational logic (if/else statements)
- Source code is generated from NFA representation
 - Often tens of thousands of lines

Algorithm 3: OpenCL-Based NFA Kernel Accepting $[Or]zo$

Input : The data stream of characters, *stream*
Match: Indication of a match found, *match*

```
1 copy(src, dest);
2 clear(states);
3 /* State storage
4 curr [4];
5 next [4];
6 match;
7 foreach c in stream do
8     /* Comparators
9     bool char_class1 = c == 'O' || c == 'r';
10    bool cmp_z = c == 'z';
11    bool cmp_o = c == 'o';
12    /* Transitions
13    if char_class1 is true then
14        | next[1] = active;
15    end if
16    if curr[1] is active and cmp_z is true then
17        | next[2] = active;
18    end if
19    if curr[2] is active and cmp_o is true then
20        | next[3] = active;
21    end if
22    if curr[3] is active then
23        | match = found;
24    end if
25    copy(next, curr);
26    clear(next);
27 end foreach
```

States

Transitions

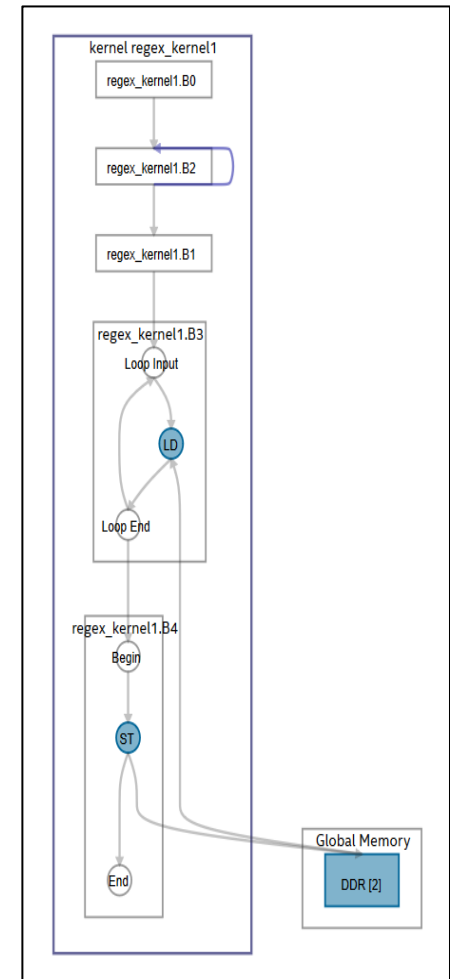
HLS Kernels – OpenCL

- Single work-item kernel
 - Recommended for FPGA development
- Accelerator fed by burst-coalesced load-store unit
 - Load multiple sequential addresses to hide DDR latency

Compile

OpenCL
Kernel

Optimize



HLS Kernels – Initial performance

- Achieves initiation interval (II) of ~1
- Infers shift registers for simple repetition

Loops Analysis				<input checked="" type="checkbox"/> Show fully unrolled loops
	Pipelined	II	Speculated iterations	Details
Kernel: regex_kernel1 (match_packet_kernel_cmp1.ct:6)				Single work-item execution
regex_kernel1.B2 (match_packet_kernel_cmp1.ct:101)	Yes	1	3	
regex_kernel1.B3 (match_packet_kernel_cmp1.ct:68717)	Yes	~1	n/a	
Fully unrolled loop (match_packet_kernel_cmp1.ct:68720)	n/a	n/a	n/a	Unrolled by #pragma unroll

	OpenCL
Maximum Frequency (Mhz)	293
Throughput (Gbps)	2.3
Look-up Tables (LUTs)	102993

Outline

Goals, Motivations, Challenges

Background

Kernel Approach

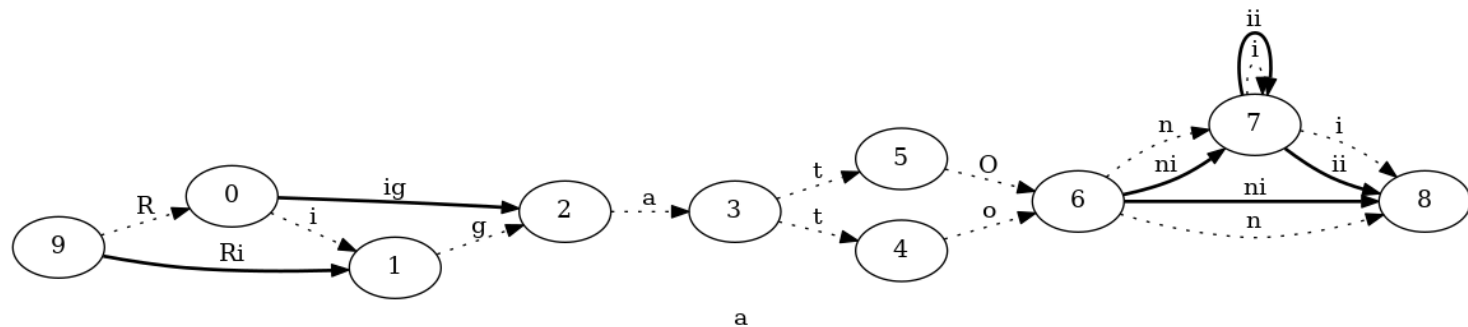
Bandwidth Scaling

Results

Conclusions, Future Work

Generating NFA – Multi-character NFA

- Input bandwidth can be increased by modifying the NFA to take multiple input characters for each transition
- The transitions in and out of each state are concatenated together to double input bandwidth



Outline

Goals, Motivations, Challenges

Background

Kernel Approach

Bandwidth Scaling

Results

Conclusions, Future Work

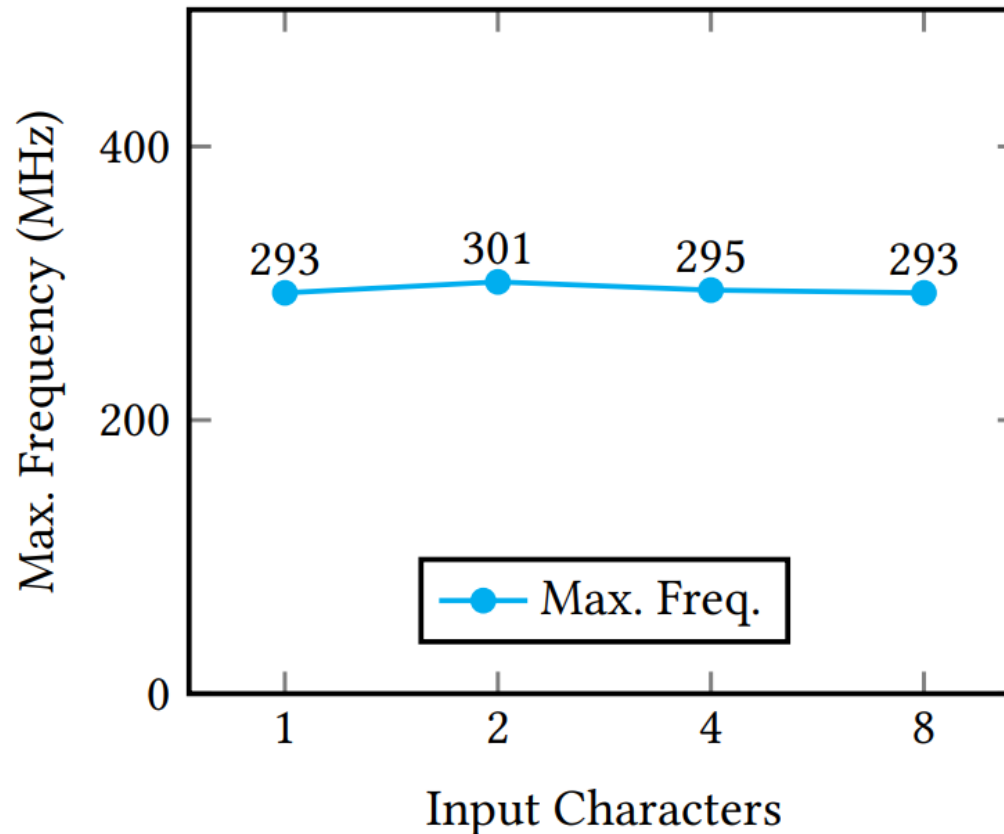
OpenCL Kernels - Testbed

- OpenCL on Arria 10
 - OpenCL SDK version – 19.4
 - Arria 10 LUTs – 854,400
 - Arria 10 manufacturing node – 20nm



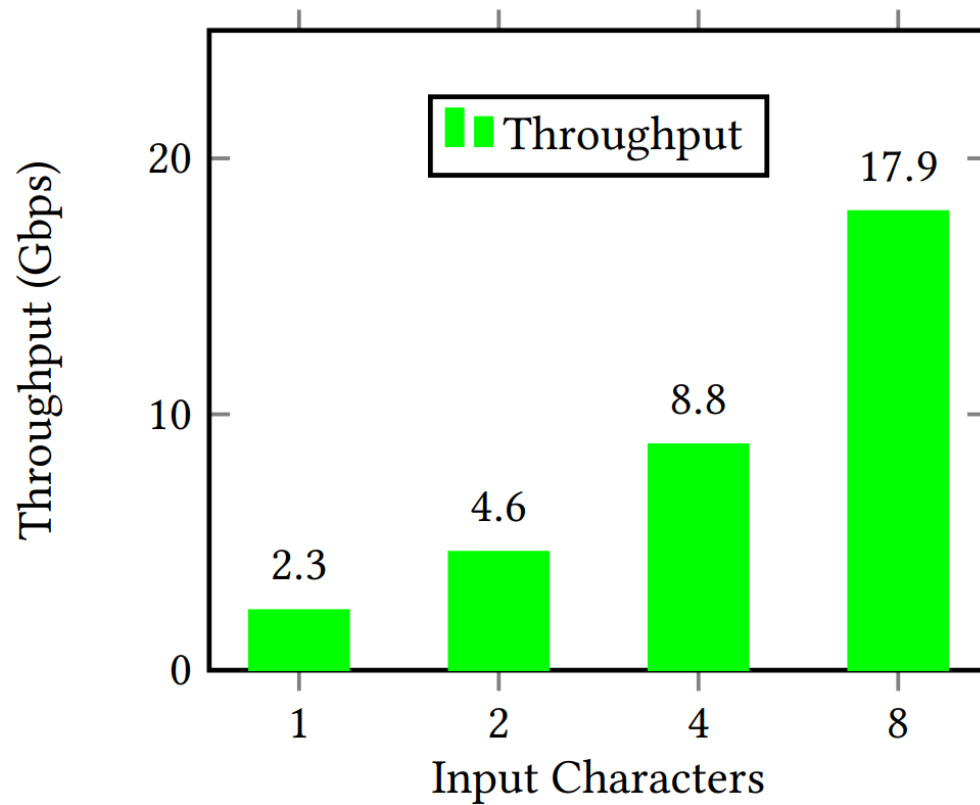
OpenCL Kernels - Performance

- Kernel Maximum Frequency



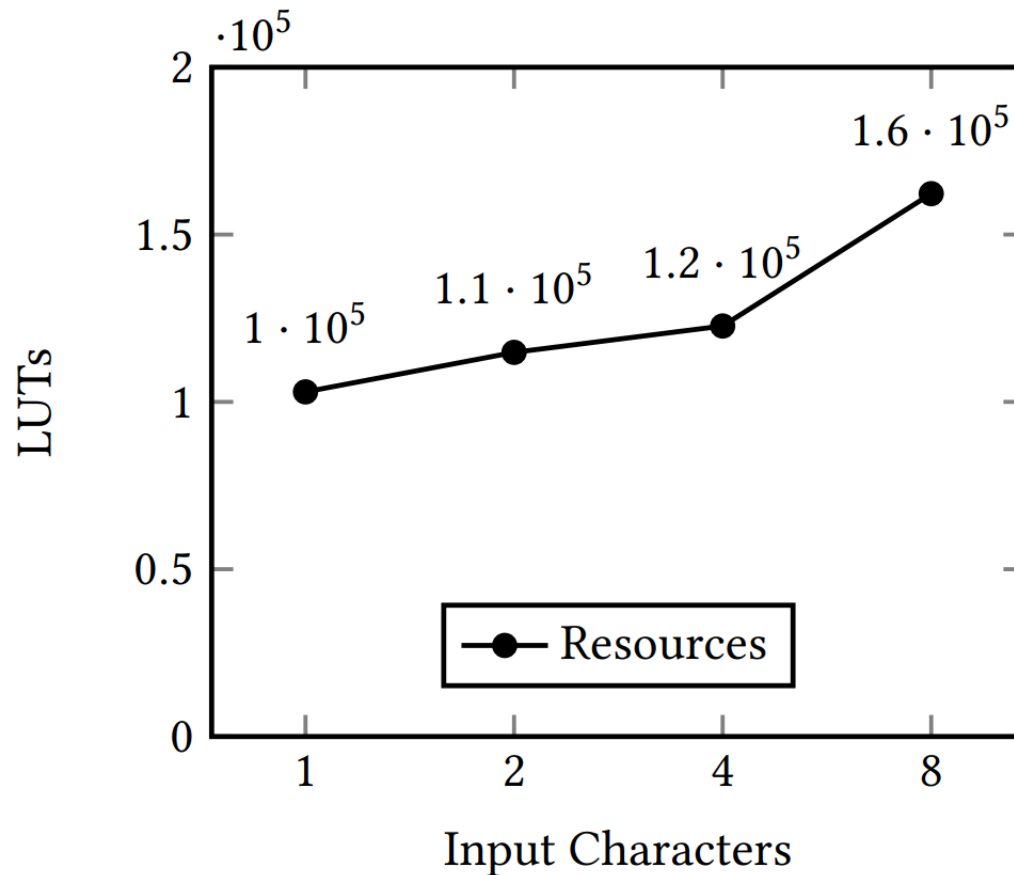
OpenCL Kernels - Performance

- Kernel Throughput



OpenCL Kernels - Performance

- LUT usage



OpenCL Kernels - Performance

■ Comparison to Prior Work

Design	Input Bandwidth (bytes)	States/ Non-meta Chars	LUT/State	Max. Frequency (MHz)	Throughput (Gbps)	Throughput Efficiency
Yang, Prasanna [24]	2	120000	.69	216	3.47	5.03
Yang, Prasanna [24]	8	120000	1.02	160.9	10.3	10.1
Yamagaki, Sidhu [23]	2	7803	.81	184	2.95	3.63
Yamagaki, Sidhu [23]	8	7803	2.51	84.42	5.4	2.15
Clark, Schimmel [7]	4	17573	3.13	219	7.00	2.24
Clark, Schimmel [7]	8	17573	5.31	114.2	7.31	1.38
Our Design	2	13049	8.80	301	4.57	.519
Our Design	4	13049	9.40	295	8.77	.933
Our Design	8	13049	12.43	293	17.88	1.43

Outline

Goals, Motivations, Challenges

Background

Kernel Approach

Bandwidth Scaling

Results

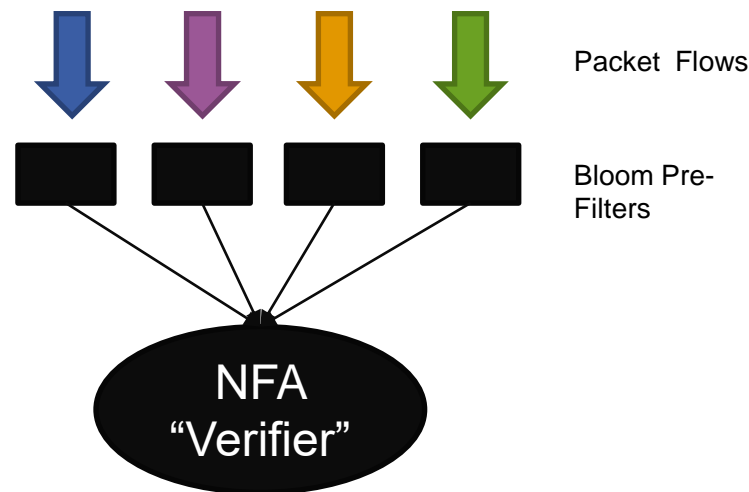
Conclusions, Future Work

Conclusions

- Good throughput scaling for HLS kernels
- Unsustainable past 8-character bandwidth
- Comparable to similar RTL solutions
 - Throughput – **best** in class among similar work
 - Resources – **worst** in class among similar work
 - Throughput Efficiency – near the bottom

Future Work

- Scale throughput past 20Gbps?
 - Data pre-filtering
 - Up to 95% of network traffic can be filtered
 - Bloom filters allow speedy lookups and compress memory needs of lookup tables



Acknowledgements

- This research was supported by SHREC industry and agency members and by the IUCRC Program of the National Science Foundation under Grant No. CNS-1738783.
- We would like to thank the support of Intel who provided access to hardware and expertise through the Intel Devcloud and the tremendous support of Kyle Buettner, Alex Johnson and Micheal Ing from NSF SHREC.

Questions