OpenCL Based Design Pattern for Line Rate Packet Processing

Jehandad Khan, Peter Athanas
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Virginia Tech
Blacksburg, Virginia
Email: {jehandad, athanas}@vt.edu

Skip Booth, John Marshall
Cisco Systems Inc.
RTP, North Carolina
Email: {ebooth, jwm}@cisco.com

***Abstract*—The ever changing nature of network technology requires a flexible platform that can change as the technology evolves. In this work, a complete networking switch designed in OpenCL is presented, identifying several high-level constructs that form the building blocks of any network application targeting FPGAs. These include the notion of an on-chip global memory and kernels constantly processing data without the intervention of the host. The use of OpenCL is motivated by the ability to rapidly change designs and to be maintainable by a wider developer community. Pieces of the design that cannot be realized using current OpenCL technology are also identified and a solution to the problem is presented.**

I. Introduction

Technologies like OpenFlow, Software Defined Networks, and Network Function Virtualization (NFV) are constantly changing the network computing landscape. Rapidly changing protocols,requirement for extensibility, demand for higher bandwidths, and increased cost sensitivity have forced both network designers and hardware vendors to rethink design methodologies. Coupled with machine virtualization, NFV adds another dimension of flexibility and complexity to the picture by using virtualized servers for network functions.

NFV attempts to address many of these problems by virtualizing many of the packet forwarding functions on standard server hardware; however, this flexibility comes at the cost of dedicated CPU cores for packet switching, higher power consumption, and other infrastructure costs required to maintain additional hardware. In contrast, dedicated switching hardware is not faced with these challenges since typical network appliances employ ASIC-based designs to achieve competitive throughput. The aforementioned challenges have put pressure on ASIC architectures to adapt, resulting in flexible architectural frameworks such as Reconfigurable Match Tables (RMT) [1] and Intel FlexPipe [2], these architectures can be configured at runtime to process different types of protocols. With the aid of Domain Specific Languages (DSLs) such as P4 [3], these architectures might be the answer to the ever changing requirements of the networking world. While such ASICs promise data rates on the order of tera-bits per second, high development costs of ASIC design coupled with lengthy design cycles make them a poor development choice for most applications. Not only do these devices require multiple years to design and develop, but once they reach maturity the requirement has moved on.

This work was supported in part by the I/UCRC Program of the National Science Foundation within the NSF Center for High-Performance Reconfigurable Computing (CHREC), Grant No. IIP-1266245.

Between general purpose CPUs and fixed function ASICs, FPGAs provide the middle ground in terms of performance, cost and flexibility. FPGAs are gradually making their way into data centers as co-processors [4] assisting CPUs in complex computations. Compared to GPUs, they offer lower power consumption, wide data paths enabling high throughput and a higher level of parallelism at a finer level of granularity. These traits make FPGAs an attractive choice for diverse applications such as machine learning [5], genome sequencing [6], high frequency trading.

The classical impediment to using FPGAs for a wider variety of applications is that they follow an ASIC-like design flow, which requires expertise in logic design, understanding of architectural design, and familiarity with place-and-route techniques for FPGA designs. Traditional FPGA design methodology also requires the use of RTL, requiring ample forethought, debugging difficulty, and long compile times, which may span multiple hours. Using a High Level Synthesis (HLS) design language such as OpenCL to target FPGAs ameliorates some of these problems by raising the level of abstraction for the designer, enabling them to express designs in a more software-friendly space. This abstraction also provides an easier debugging environment and more concise and comprehensible code. Thus, before committing to a multi-hour compile, the designer is able to verify functional correctness, potentially identifying mistakes earlier in the design cycle.

Modern OpenCL tools [7] efficiently generate RTL from a HLS specification, leaving little room for improvement in terms of area and performance as compared to the productivity advantages being gained. This trait makes OpenCL a viable design methodology for quickly creating efficient FPGA designs. However, there are certain caveats associated with this approach, such as portability issues, dependence of performance on coding style, and efficient utilization of memory resources.

In this work, design patterns are identified that enable high-performance packet processing applications on FPGA targets. A simple router/switch design is used for this exploration and

the Altera Arria 10AX115S2F45I2SGES device is used as the reference platform. The key design philosophy for this endeavor is described, drawing similarities and contrasts between ASIC-based architecture and how the flexibility afforded by FPGAs may be leveraged in this space.

The organization of this paper is as follows: In Section II the switching problem is presented, along with the switch abstraction employed and some explanation for the basis of this approach. In Section III, the building blocks for the switch architecture are introduced, in Section IV the overall switch architecture is described using the primitives introduced in the previous section, and some related work is reviewed. Section V presents some results with conclusions and recommendations at the end.

II. Abstract Switch Architecture

Fig 1 shows the abstract switch model implemented in this paper. This is a standard architecture followed by many switching ASICs, such as Reconfigurable Match Tables (RMT) [1], and the Intel FlexPipe [2]. The same general approach has been adopted by OpenFlow [8], P4 [3], and PX [9] in a more abstract model.

Fig 1 describes three building blocks of the switch data plane. The parser, matching tables, and control (ingress / egress). First, the parser parses the incoming packets into different headers, depending upon the requirements this might be a fixed parser with known protocols, or a configurable one. A general parser might be able to handle arbitrarily complex header stacks as well as recursion. Such features are more readily implemented in software as compared to hardware. Next, fields from the parsed representation are matched against tables for update decisions about either the meta-data or pieces of the packet data itself. For example, a *longest prefix match* may enable a *forwarding information base* lookup using a matching stage on the source/destination IP address, and updating either the egress port in the meta-data or some other field. Finally, the ingress and egress control fields determine the order of operations along with other features such as recirculation, quality of service, and buffer management.

Experiments in this paper are based upon the minimal realization of this architecture in the form of a Layer 3 router. It consists of a parser that parses the Ethernet and IPv4 headers, followed by two matching and update stages for the ingress control and one for the egress. For the sake of simplicity, buffer management, queuing, and other infrastructure are not implemented in this design. Apart from these stages, the switch is also equipped with an IPv4 *checksum verify and update* stage at the ingress and egress respectively. In a more complex scenario, there may be many matching stages forming a graph of packet flow. Typically, such a graph is scheduled to a fixed pipeline [2] whereas in the case of an FPGA, this is not a limitation due to the flexible nature of the device. An FPGA-based design is only constrained by device area and the increase in latency due to the depth of the pipeline. In the context of packet processing, this fact enables an HLS-based

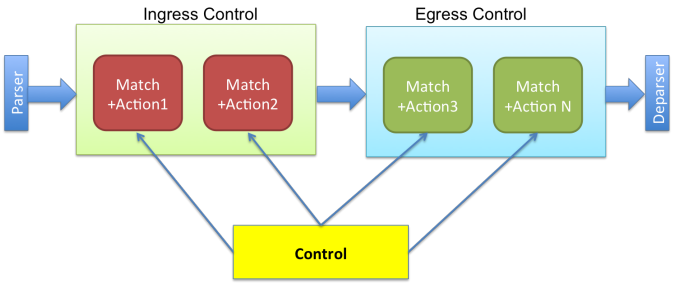

Fig. 1. Architectural Overview of a P4 Program

FPGA design to approximate CPU-based design paradigm only limited by the physical characteristics of the device.

After laying down the conceptual ground work, the next section identifies the architectural building blocks necessary for realizing this design on an FPGA.

III. HLS Framework for Packet Processing

In this study, Altera's OpenCL tool chain [10] is used to target an Arria 10 device. Lessons learned here, however, may easily be extended to other FPGA tool vendors. OpenCL enables the use of a single code base on multiple heterogeneous targets, such as GPUs and CPUs. While the syntax and code may be portable from one platform to another, performance varies widely when the same code is run on different architectures, particularly from an FPGA perspective. Not only is performance portability a myth between different device architectures, but even within the FPGA world, performance and utilization results vary widely when the same code is compiled using tool chains from different vendors. This variability may be attributed to the major differences in device architectures and to the many ways in which the same OpenCL code may be mapped to an FPGA target.

Motivated by this variability, OpenCL-based design patterns are identified that would give a high degree of parallelism at a moderate cost of device resources. Much like software design patterns and other benchmarks [11], these building blocks can be combined to build larger and more complex systems. Moreover, lessons learned here may also be incorporated into an automated system to generate efficient code.

The current OpenCL standard assumes that a kernel is invoked with a pointer to a work load, and the kernel operates on the given workload until it is finished. Once the work is done, results are copied back to the host. The paradigm is unsuitable for any low-latency, high-bandwidth streaming application such as packet processing. In addition, many streaming applications effectively run forever on these streams of data, and breaks the mold of traditional CPU-driven kernel launches. For example, the target platform is equipped with an on-board Ethernet interface that might be feeding data in to the design. This problem is exacerbated by the fact that the memory controllers in the device might optimize writes centered around the kernel completion event; therefore, an attempt to read data back from a kernel before it has terminated might either give inconsistent results or fail in some other way.

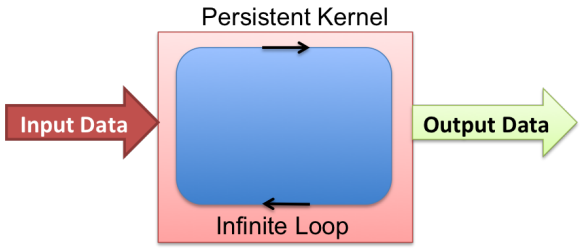

Fig. 2. A persistent Kernel

To solve this problem, a construct called *persistent kernels* is introduced, as described in the next section.

A. Persistent Kernels

A kernel that does not terminate once launched by the host is a persistent kernel. Data items are fed to this kernel using OpenCL pipes, it consists of an infinite loop, and each iteration of the loop waits on the input channel. Once a valid data item is available, the kernel code operates on the data item and writes the result to the output channel. Since Altera OpenCL (AOCL) does not support OpenCL pipes in simulation, Altera Channels have been used throughout this study to realize this functionality. Fig 2 depicts the structure of a persistent kernel. The arrows indicate incoming and outgoing channels carrying data, and the box in the center represents the kernel with an infinite loop. If a particular kernel needs input from more than one kernel, the persistent kernel might poll multiple channels in a non-blocking manner and act upon data whenever it is available. Channels are implemented in hardware as FIFOs making them relatively inexpensive and abundantly available. This construct enables designers to express a greater degree of parallelism. For example, in OpenCL parallelism cannot be expressed within a single kernel. Even if the sections of the code are completely independent, they would still be part of a single pipeline. Moreover, a single kernel is optimized as a whole; thus, if one section of the code stalls, the whole kernel is stalled. In the present scenario, this implies that if a table lookup is stalled and the parser is a part of the same kernel, the entire operation would stall even if the two are completely independent. This fact has serious performance implications.

The notion of persistent kernels is not new and has been explored previously in a slightly different contexts on GPUs [12] as *persistent threads*. There are key differences however: the processing model in this paper is task based; therefore, thread scheduling is not a concern. Moreover, persistence in the context of this design implies a kernel that does not terminate once it is launched. The persistent kernel model enables the expression of this independence and parallelism. In a packet processing scenario, each packet is handled independently; therefore, the ability to exploit parallelism is of paramount importance. Large number of packets may be kept in flight using this technique, which is necessary for a high-throughput design.

B. Global On-chip Memory

To achieve line-rate performance, many packets have to be kept in flight. Variable latency or low throughput in accessing this data may lead to packet loss and jitter. Off-chip memory systems such as DDR memory controllers cannot provide such performance given the memory operations per second required for packet processing. Large cache lines of traditional memory systems are unsuitable for packet editing operations. Packet editing requires access to a few bytes at a time only. Memory controllers are also prone to stalling, and nondeterministic delays makes them a poor choice for storing in-flight packet data.

AOCL does not support lgobal on-chip memory, which would be ideal for this scenario since this data would be shared among different persistent kernels, and passing the entire packet from one kernel to another would be resource intensive. To solve this problem, a persistent kernel that declares local storage arrays is employed. The AOCL compiler realizes the local memory as an M20K block on the Arria 10 device. Each M20K block has two input and two output ports. This limitation requires that no more than two persistent kernels should write or read from the memory server. Otherwise, arbitration logic would be required. This is not desirable for performance and resource utilization reasons. Global on-chip memory is supported by Xilinx SDAccel [13]; however, the developer has little control over its parameters.

IV. Overall Switch Architecture

Using the building blocks outlined in the previous section, the abstract switch architecture discussed in Section II is mapped using OpenCL. Fig 6 shows the complete router realization. The ingress kernel receives the packet stream either from the host in DMA chunks or from an on-board Ethernet interface. It constructs a Packet Header Vector (PHV) with the available information, and passes it down the processing pipeline. Likewise, the de-parser kernel writes the packet to the main memory or to the I/O channel for the on board Ethernet channel.

Similar to the RMT architecture [1], the PHV containing the parsed-packet representation along with meta-data is transferred from one stage to another. However, choosing FPGA as the target allows the flexibility of mapping as many stages to the device – only limited by the physical resources of the target device. Also, the width of the PHV may be tuned to trade-off performance and utilization.

A. Parser Kernel

The parser receives the PHV with basic meta-data, such as ingress port and packet length. It also receives the address of the packet in the memory server described in Section III-B. In this context, the memory server might be referred to as the packet server. Since an FPGA-based OpenCL design has a theoretical maximum achievable clock frequency of no more than 230 to 240 MHz, wide data paths are employed to process two packets per clock cycle; therefore, the parser kernel reads two PHVs from the input channel and directs

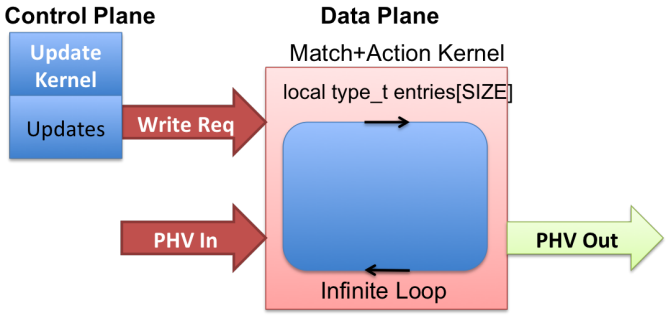

Fig. 3. Match Action Stage Architecture

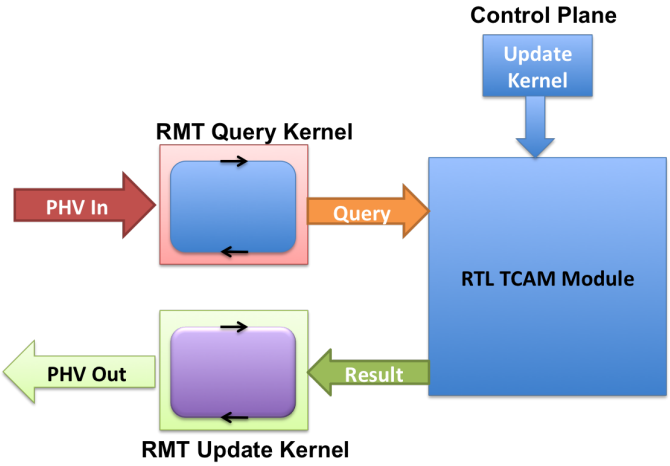

Fig. 4. RTL based Match Action Stage Architecture

them to the parser loop. This loop is fully pipelined by the AOCL compiler. Once fully populated, it provides two parsed-packet representations every clock cycle.

B. Matching Stages

PHV from the parsed packet is passed onto the next matching kernel for lookup and update operation. Two types of matching kernels were implemented: TCAM-based ternary match, and exact match. The aim is to create a minimal number of TCAM entries on chip, which may later be extended to a more sophisticated architecture enabling a larger number of entries using off-chip memory if necessary.

Initially the TCAM entries were realized as local memory in the RMT kernels, similar to the packet server architecture. The RMT kernel would listen on the upstream stage (either the parser or previous match stage) for the PHV data. Once a new data item was available, the PHV would be read from the FIFO, matched to the local array and the associated action data acted upon. The updated PHV would then be written to the downstream channel.

Control Plane: In a traditional switch architecture, population of the tables is managed by the *control plane*. In this architecture, each RMT stage kernel has an associated control kernel to add entries to the table. These kernels are launched by the host to add or update table entries. The TCAM kernel would read the data items and update its local memory array. This architecture is depicted in Fig 3.

Creating a large array of entries being evaluated in parallel uses an unaffordable amount of logic since the compiler would either realize the array completely as registers to facilitate parallelism, or in some cases depending upon the OpenCL code structure, as a large number of shallow BRAMs. Either method is wasteful of logic resources. The exercise highlighted the limitations associated with HLS-based design expression. Moving the TCAM to an RTL-based design improved the device utilization and performance. This decision was motivated by the following key factors: a high-performance TCAM is an essential part of any switch, the TCAM architecture is not expected to change from one P4 design to another; therefore, a static RTL module is acceptable. An RTL-based design would allow more control and precision around performance and area decisions. Before investing the time and effort in designing and integrating an RTL-based TCAM with an OpenCL pipeline, the maximum achievable throughput in an OpenCL-based design was determined. For the design at hand with a single parser/de-parser stage and three RMT stages, a throughput of 70 Million Packets Per Second (Mpps) was achieved, which indicates the theoretical maximum performance achievable.

C. RTL Based TCAM

AOCL allows the integration of custom RTL into OpenCL designs by making the RTL module a part of the OpenCL Board Support Package (BSP). The module must expose Altera Avalon streaming interfaces for input and output data. These channels are exposed inside the OpenCL domain as I/O channels that the OpenCL design can write to and read from like any other channel. Since the TCAM design is not the objective of this study, a naive TCAM implementation is used that sequentially reads through data stored in BRAM cells in a pipelined manner. This decision was taken to concentrate on the OpenCL aspects of the system, ignoring the details of TCAM design. Also, a smart caching algorithm around the TCAM [14] may relax the requirement for a large TCAM. Another component similar in interface to the TCAM module was designed to achieve exact match classification. Fig 4 shows the RTL TCAM in conjunction with the OpenCL kernels. Together, these components make up an OpenCL-based RMT stage. Similar to the OpenCL-only RMT stage, there is a control kernel marked *update kernel*; however, the match and action parts have been split to avoid stalls in the action kernel and keep more requests in flight. The query kernel may be fused with the action/result kernel of the previous stage to avoid redundant kernels.

This process also highlighted the limitations of HLS, demonstrating that not all types of logic may be optimally synthesized. It may also be pointed out that despite its shortcomings, OpenCL enables great productivity gains by supplying control-plane logic, pipelining and other primitives; thus, being forced to write a small part of the design in RTL is still better than to write the entire design in RTL.

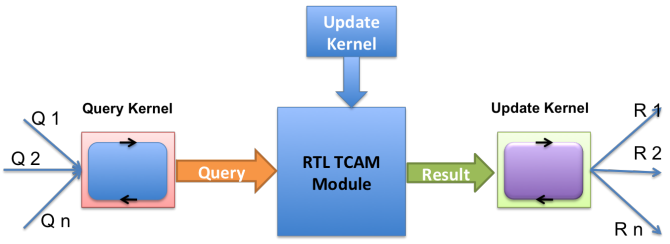

Fig. 5. Resource Sharing of TCAM in Multiple Stages

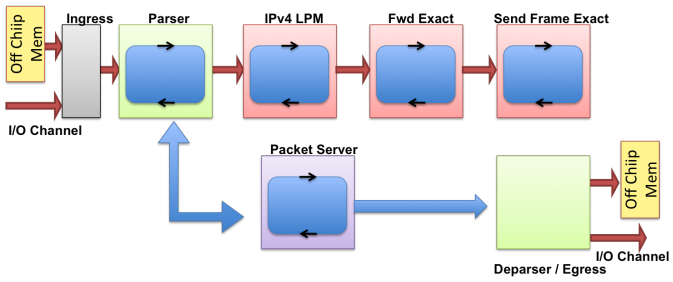

Fig. 6. All the kernel required to implement the simple router

D. TCAM Resource Sharing

If resources are scarce, a single RTL TCAM may be shared among multiple RMT stages as depicted in Fig 5. The RMT stage, instead of writing to the TCAM query channel, now directly writes to the query kernel that listens to all RMT stages. Each query request is tagged with the incoming RMT stage's ID to be passed along by the RTL TCAM with the lookup result. This ID enables the result kernel to route the outcome of the lookup to the appropriate kernel.

E. Related Work

While the focus of this work has been to design a *complete* switching architecture in OpenCL for FPGA targets, different attempts have been made at different parts of the switching problem for example, Attig et. al [15] implement a 400 Gb/s parser and Jiang et. al [16] present matching engines based on FPGA. Naous et al. [17] have implemented an OpenFlow compliant switch on the popular NetFPGA platform utilizing a RTL-based design methodology. Brebner et. al in [18] present an automatic architecture which can reach a throughput of 200 Mpps with a TCAM size similar to this work and appropriate parameter choice, however the work presented here synthesizes router specification described in OpenCL as compared to a fixed logic architecture as in [18]. There also have been attempts at using GPUs to switch packets, such as [19], [20] and [21]. Han et. al [19] and Li et. al [21] present GPU based implementations of routers. GPUs have high throughput compared to FPGAs in terms of data processing; however, the latency introduced by GPUs in packet processing makes them unsuitable for line rate processing. Moreover, high power requirements hamper their deployment in edge network scenarios. GPUs are also hampered by the unavailability of on-board interface to operate independently. All of these are important considerations for network appliances, making GPUs less favorable for network processing. Makkes et. al [22] in their recent work present a high performance lookup engine on GPUs and CPUs, this engine batches up lookup requests to exploit GPU thread size, it also makes certain assumptions around the number of prefixes and the structure of the data in general. Rinta-Aho et. al [23] and Nikander et. al [24] present work in compiling router specification described in Click [25] to an FPGA based router. While interesting in it's own right, attempting to compile an entire C code base to FPGA [23] does not produce optimum results for performance. We compare work presented here to [23] in the results section. Kim et. al [20] propose a router named Network Balancing Act (NBA) which achieves comparable throughput to this work on a heterogeneous platform, their presented work has features including IPSec and Intrusion Detection System. ReClick router by Unnikrishnan et. al [26] present another attempt at converting a Click router specification in to HDL based FPGA design. The authors present a feature rich implementation however, the throughput achieved is not competitive.

V. Results

All experiments were carried out on the Arria 10 GX FPGA Development Kit with 4 GB of DDR4 RAM. The board was plugged into a Cisco UCS C240 M4 server with two Intel Xeon E5-2667 v3 CPUs running at 3.2 GHz with 64 GB of RAM. The design was compiled using Altera OpenCL SDK Version 15.1. The custom TCAM and matching engine were integrated in the design by modifying the Altera supplied BSP.

To determine realistic performance and area expectations, the resource utilization and performance metrics of individual components are examined. Table I shows the utilization and performance metrics for the look up engine. The numbers in parenthesis are the resource utilization for the complete design along with the launching kernels to test the design. In this relatively simple design, each M20K block holds eight data/mask pairs that are searched sequentially. The TCAM in the current design is 40-bits wide.

Table II depicts the resource utilization in different parts of the design, indicating the efficiency of the design paradigm. It is evident from tables II and III that much of the device is occupied with the TCAM engine. This can partly be attributed to the naive TCAM design as stated earlier. To achieve maximum performance, a dedicated lookup engine was instantiated for each match+action stage, namely one *longest prefix match* stage and *exact match* engines for the *send frame* and *forwarding* stages. The TCAM engine, being the slowest component in the chain, restricted the performance of the overall system to a little under 40 Mpps for 64-byte packets, with the limiting factor being the naive TCAM design. Without this limitation, the design is able to process at 70 Mpps.

If resources are constrained, the result kernel and the query kernel for the subsequent RMT stage may be merged to conserve resources. Table IV indicates the impact of this optimization on the overall device utilization. The first column

TABLE I
LOOKUP ENGINE SPECS

	TCAM Engine	Exact Match Engine
Logic Utilization (ALMs)	60,959 (90,538)	58,628 (89,287)
Dedicated Logic Registers	138,477 (196,859)	144,549 (208,861)
RAM Blocks	65	129
Block Memory Bits	82,432 (853,746)	82,432 (909,554)
Fmax	242 MHz	241 MHz
Lookups Per Second	45 Million	57 Million
Overall Logic Utilization	21%	21%

TABLE II
DEVICE UTILIZATION OF DIFFERENT SWITCH COMPONENTS

	IPv4 Checksum	Parser	Packet Server	De-parser
Logic Utilization (ALMs)	2068	4929	4012	9870
Logic Registers	5325	10835	13008	25968
RAM Blocks	22	0	40	35584
Block Memory Bits	3806	0	200704	14

TABLE III
MATCH ACTION COMPONENTS UTILIZATION

	Add	Query	Result
Logic Utilization (ALMs)	1021	1254	1503
Dedicated Logic Registers	2123	2710	3566
RAM Blocks	2	0	0
Block Memory Bits	6784	0	0

TABLE IV
ROUTER RESOURCE UTILIZATION

	Merged Kernels	Separate Kernels
Logic Utilization (ALMs)	257,473 (60%)	264,736 (62%)
Dedicated Logic Registers	590,946	603,921
RAM Blocks	1139 (42%)	1211 (45%)
Block Memory Bits	3,818,190 (7%)	3,908,846 (7%)

represents the design with merged kernels, and the second column indicates the device utilization without the merge kernel optimization. The numbers in the parenthesis are the percent utilization of the overall device resources. It may be noted that the resource utilization includes all of the auxiliary logic needed by the design, such as the DDR controller and the soft logic to interface with the PCIe interface. Due to the scarcity of data on a FPGA-based router performance, it is difficult to make a fair comparison. To emphasize the accomplishment of this work, Table V compares the performance of the current OpenCL-based design to other design approaches. It might be noted that performance data for more recent FPGA designs is not available; therefore, the comparison presented in Table V cannot be used to draw a fruitful comparison.

TABLE V
PERFORMANCE COMPARISON

	Packets Per Sec	Platform
NetFPGA (Ref)	415 K	Xilinx Virtex II Pro
NetFPGA (OpenFlow)	1.95 M	Xilinx Virtex II Pro
Click2NetFPGA	178 K	Xilinx Virtex II Pro
OpenCL Router	40 M	Arria 10 GX 115

VI. CONCLUSION

OpenCL is fast becoming a viable language of choice for FPGA development. This work presents a design paradigm using OpenCL for packet processing applications on FPGA targets. While it may still be difficult to fully express all intricacies of hardware design using OpenCL, the productivity benefits of OpenCL make it a worthwhile investigation. Some design primitives targeting streaming applications in general and packet processing applications in particular were also presented. Some of the suggested future work includes determining a solution to the performance bottleneck associated with the TCAM engine using a more sophisticated implementation. More features such as queue management and buffering should be explored as important avenues of investigation.

REFERENCES

[1] P. Bosshart, G. Gibb, H.-s. Kim, G. Varghese, N. McKeown, M. Izzard, F. Mujica, and M. Horowitz, "Forwarding Metamorphosis: Fast Programmable Match-Action Processing in Hardware for SDN," *ACM SIGCOMM Computer Communication Review*, pp. 99–110, 2013.

[2] L. Jose, L. Yan, G. Varghese, and N. McKeown, "Compiling Packet Programs to Reconfigurable Switches," *12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15)*, pp. 103–115, 2015.

[3] P. Bosshart, G. Varghese, D. Walker, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, and A. Vahdat, "P4: Programming Protocol-Independent Packet Processors," *ACM SIGCOMM Computer Communication Review*, vol. 44, no. 3, pp. 87–95, 2014.

[4] L. Ling, J. Grecco, H. Mitchel, L. Dong, P. Gupta, N. Oliver, C. Bhushan, W. Qigang, A. Chen, S. Wenbo, Y. Zhihong, A. Sheiman, and I. McCallum, "High-performance, energy-efficient platforms using in-socket FPGA accelerators," in *Proceeding of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '09*. New York, New York, USA: ACM Press, feb 2009, p. 261.

[5] K. Ovtcharov, O. Ruwase, J.-y. Kim, J. Fowers, K. Strauss, and E. S. Chung, "Accelerating Deep Convolutional Neural Networks Using Specialized Hardware," *Microsoft Research Whitepaper*, pp. 3–6, 2015.

[6] I. Li, W. Shum, and K. Truong, "160-fold acceleration of the Smith-Waterman algorithm using a field programmable gate array (FPGA)," *BMC bioinformatics*, 2007.

[7] D. P. Singh, T. S. Czajkowski, and A. Ling, "Harnessing the power of FPGAs using Altera's OpenCL compiler," in *Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays - FPGA '13*. New York, New York, USA: ACM Press, feb 2013, p. 5.

[8] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, p. 69, mar 2008.

[9] G. Brebner and W. Jiang, "High-speed packet processing using reconfigurable computing," *IEEE Micro*, vol. 34, no. 1, pp. 8–18, 2014.

[10] Altera Corporation, "Altera SDK for OpenCL," 2015. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf

[11] K. Krommydas, W. C. Feng, M. Owaida, C. D. Antonopoulos, and N. Bellas, "On the characterization of OpenCL dwarfs on fixed and reconfigurable platforms," *Proceedings of the International Conference on Application-Specific Systems, Architectures and Processors*, pp. 153–160, 2014.

[12] K. Gupta, J. A. Stuart, and J. D. Owens, "A study of persistent threads style GPU programming for GPGPU workloads," in *2012 Innovative Parallel Computing, InPar 2012*. IEEE, may 2012, pp. 1–14.

[13] Xilinix Corporation, "SDAccel Development Environment." [Online]. Available: http://www.xilinx.com/products/design-tools/software-zone/sdaccel.html

[14] N. Katta, O. Alipourfard, J. Rexford, and D. Walker, "CacheFlow : Dependency-Aware Rule-Caching for Software-Defined Networks Categories and Subject Descriptors," in *ACM Symposium on SDN Research*, 2016.

[15] M. Attig and G. Brebner, "400 Gb/s programmable packet parsing on a single FPGA," *Proceedings - 2011 7th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2011*, vol. 400, pp. 12–23, 2011.

[16] W. Jiang and V. K. Prasanna, "Scalable Packet Classification on FPGA," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 9, pp. 1668–1680, sep 2012.

[17] J. Naous, D. Erickson, G. A. Covington, G. Appenzeller, and N. McKeown, "Implementing an OpenFlow switch on the NetFPGA platform," in *Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems - ANCS '08*. New York, New York, USA: ACM Press, Nov 2008, p. 1.

[18] G. Brebner and W. Jiang, "High-speed packet processing using reconfigurable computing," *IEEE Micro*, vol. 34, pp. 8–18, 2014.

[19] S. Han, K. Jang, K. Park, and S. Moon, "PacketShader," *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM - SIGCOMM '10*, vol. 40, no. 4, p. 195, 2010.

[20] J. Kim, K. Jang, K. Lee, S. Ma, K. Shim, and S. Moon, "NBA (network balancing act): A high-performance packet processing framework for heterogeneous processors," *European Conference on Computer Systems (EuroSys)*, 2015.

[21] P. Li and Y. Luo, "P4GPU," in *Proceedings of the 2016 Symposium on Architectures for Networking and Communications Systems - ANCS '16*. New York, New York, USA: ACM Press, mar 2016, pp. 125–126.

[22] M. X. Makkes, A. Varbanescu, C. de Laat, and R. Meijer, "Fast packet forwarding engine based on software circuits," *Proceedings of the 12th ACM International Conference on Computing Frontiers - CF '15*, pp. 1–8, 2015.

[23] T. Rinta-Aho, M. Karlstedt, and M. P. Desai, "The Click2NetFPGA toolchain," *USENIX Annual Technical Conference (USENIX ATC 12)*, p. 7, Jun 2012.

[24] B. Pekka Nikander, S. D. Sahasrabuddhe, and J. Kempf, "Towards Software-defined Silicon: Experiences in Compiling Click to NetFPGA," 2010.

[25] E. Kohler, R. Morris, B. Chen, J. Jannotti, and M. F. Kaashoek, "The click modular router," *ACM Transactions on Computer Systems*, vol. 18, no. 212, pp. 263–297, aug 2000.

[26] D. Unnikrishnan, J. Lu, L. Gao, and R. Tessier, "ReClick - A modular dataplane design framework for FPGA-based network virtualization," in *Proceedings - 2011 7th ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2011*. IEEE, oct 2011, pp. 145–155.