An OpenCL Framework for Distributed Apps on a Multidimensional Network of FPGAs

Abhijeet Lawande*, Alan D. George†, Herman Lam‡
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL 32611
*lawande@chrec.org, †george@chrec.org, ‡lam@chrec.org

***Abstract*—In an effort to offset the rapidly increasing data volume processed by large data centers today, their architects have increasingly been exploring unconventional architectures like FPGAs. Large-scale RC systems like Novo-G# show promise for both big-data processing and HPC, but are limited by a lengthy and difficult design process. In this paper we present a mixed MPI/OpenCL framework that enables rapid and simple multi-FPGA app development on Novo-G# with support for multidimensional inter-FPGA communication. The framework encapsulates inter-FPGA links into Altera OpenCL channels, abstracting away many of the complexities of inter-FPGA communication, and achieves an aggregate data rate of 288 Gbps per FPGA over six input and six output links. We use case studies and analysis to showcase a methodology for efficient design of multi-FPGA OpenCL apps on Novo-G# with our framework, and demonstrate its use to create various multi-FPGA applications.**

I. Introduction

With the continuing and escalating demands from big data and extreme-scale computation as we head towards Exascale, there is an increasing interest in the research community to study and deploy scalable FPGA-based reconfigurable computing (RC) systems. Additionally, major companies like Microsoft [1], Baidu [2], and Intel [3] have already started integrating FPGAs in their systems and cloud solutions. A notable example of such a system from Microsoft is Catapult [1], which consists of 1632 mid-sized FPGAs interconnected in independent 6x8 tori, and has proven to be effective in accelerating non-traditional applications such as page ranking.

Novo-G [4], [5], a scalable RC system of 52 servers containing 400+ FPGAs was developed at the University of Florida and was successful in accelerating impactful scientific applications in the areas of bioinformatics, computational finance, image and signal processing, and others [6]. Novo-G# is an enhancement of Novo-G which was developed for acceleration of communication-intensive apps. The system features 64 ProceV boards from Gidel, each equipped with a Stratix V D8 FPGA, and the boards are connected with high-speed links in a 3D torus network (4×4×4).

Although successful in accelerating the targeted applications, developing large-scale RC apps for systems such as Catapult and Novo-G# is still a daunting task. Factors such as the long turnaround time of FPGA designs, high barrier to entry for RTL development, and lack of design tools for multi-FPGA apps greatly reduce the usability and wide acceptability of such systems in the HPC community.

Altera OpenCL (AOCL), a recently released tool from Altera, aims to mitigate part of FPGA design cycle problem. It supports a subset of the OpenCL 2.0 specification and adds vendor extensions useful for FPGA development. The AOCL channel extension, for example, currently allows point-to-point communication between OpenCL kernels on the same FPGA. Moreover, AOCL provides not just the OpenCL-to-RTL compiler, but an end-to-end framework including a host runtime library, software emulation support, profiling tools, and reduced hardware compilation times.

In this paper, we present a multi-FPGA OpenCL framework for Novo-G# that simplifies and speeds up the development of multi-FPGA, communication-intensive apps using AOCL. With this framework we add new capability to the Novo-G# system in the form of support for OpenCL-based distributed-app development and execution. Further, we extend the definition of AOCL channels to allow AOCL kernels running on different FPGAs to communicate with each other directly. Finally, we showcase three multi-FPGA case studies on Novo-G# to demonstrate and evaluate our framework. "Ping-pong" is a benchmark run on two FPGAs that enables us to analyze efficient channel usage and achieves an effective data rate of 24 Gbps on every inter-FPGA channel. "Channelizer" showcases the simplicity with which a single-FPGA AOCL app with multiple kernels can be decomposed to run across two FPGAs without loss of performance. "2DFFT" demonstrates how AOCL apps can be scaled effectively onto multiple AOCL devices across the system with a speedup of 6.9 on eight FPGAs versus a single-FPGA baseline.

In short, we have created a multi-FPGA OpenCL framework for Novo-G# that aims to simplify multi-FPGA app development by reducing the barrier-to-entry through High-Level Synthesis (HLS), shortening the turnaround time for multi-FPGA designs, and providing an end-to-end tool flow. The use of such a framework goes a long way towards improving the case for the use of large-scale RC systems in data centers and HPC.

The remainder of this paper is organized as follows. Section II describes the Novo-G# machine and AOCL in detail. Section III presents the multi-FPGA OpenCL framework, and explains how inter-FPGA communication is integrated into

978-1-5090-3867-1/16 $31.00 © 2016 IEEE
DOI 10.1109/IA3.2016.12

AOCL and how the framework is extended to multiple devices and servers. Section IV describes the three case studies, details of their implementation on multiple FPGAs, experiments run on Novo-G#, and any observations of note. Finally, Section V presents our conclusions and future plans for our framework.

II. Background

In this section, we briefly describe the Novo-G# system that our work targets, and the work that has previously been done to support multi-FPGA development on it. We also introduce AOCL, its capabilities, and how it compares to other HLS efforts in the past.

A. Novo-G#

Novo-G# [6] is an expansion to the Novo-G [5] reconfigurable supercomputer that focuses on accelerating communication-intensive multi-FPGA apps. Traditionally, such communication makes use of centralized networks such as InfiniBand and requires multiple transfers between the FPGA and the host. Novo-G#, however, provides a high-bandwidth, low-latency 3D-torus network that allows the FPGAs to communicate directly with each other. The FPGA accelerators used are ProceV boards from Gidel [7], with a single high-density Stratix V D8 device containing 262k adaptive logic modules each, and that can drive inter-FPGA links at 40 Gbps per link.

Each Novo-G# server contains four ProceV boards in a single chassis and is connected to the host CPUs over a PCI Express 3.0 x8 bus. The servers themselves communicate with each other, and with a separate headnode, over Gigabit Ethernet and QDR InfiniBand. Currently, Novo-G# [6] stands at 64 ProceV nodes connected in a $4\times4\times4$ torus in 16 servers, with an upgrade of 64 additional nodes soon to be completed.

Over the last two years, we have built a flexible protocol stack and router for Novo-G# , which allows app code running on an FPGA to send packets to any node within the 3D torus network [6]. The protocol stack supports several different physical and data-link layer protocols, one of which is the Interlaken PHY IP from Altera [8]. The Interlaken protocol is of special interest to this work since it provides many of the key physical and data-link layer services in silicon, such and 64b/67b line coding, automatic clock recovery and CRC32 error detection. However, the restriction of using HDL and custom network interfaces has limited Novo-G#'s use by researchers without hardware expertise.

B. Altera OpenCL

Altera OpenCL was officially released with version 13.1 of the Quartus II development tool in November 2013. AOCL provides a compiler compliant with partial support for version 2.0 of the OpenCL specification, as well as tools for functional emulation and profiling of OpenCL code. For execution on the hardware, AOCL provides support for single-command offline compilation of the device kernels, and a host runtime library to instantiate and control kernel execution on the FPGA. To allow for variations in board design and IP, the hardware interfacing code and low-level PCI Express driver are provided by the board vendor in the form of a Board Support Package (BSP). Despite being a relatively recent development, AOCL has already been used successfully in the development and acceleration of FPGA apps as described in [9]–[11]

Unlike the C-based HLS tools described in [12]–[15], OpenCL is an explicitly parallel language, allowing it to map well onto the FPGA architecture. OpenCL organizes data elements into work-items, and data processing into kernels, each with the code for processing a single work-item. In this manner, an OpenCL app may be organized as a single large kernel sequentially processing all work-items (the C model), or a series of multiple small kernels, each processing a number of work-items in parallel (the CUDA model).

Tools and methods for converting OpenCL such as OpenRCL [16], SOpenCL [17], and [18], or from CUDA [19] to RTL code do exist; however they are limited only to the generation of the datapath. In contrast, Altera OpenCL provides an end-to-end toolchain that is intended to be used from design space exploration, to compilation, to execution. However, this toolchain has been limited to single-FPGA OpenCL app development thus far.

III. Approach

In this section, we describe our methodology for integrating the existing Novo-G# protocol stack with the multi-FPGA OpenCL framework, enabling the use of inter-FPGA channels within OpenCL. We also describe how an AOCL app can be scaled to run on multiple devices and multiple servers.

A. Inter-FPGA communication support

Comuunication between FPGAs on Novo-G# is handled through their in-house protocol stack that provides physical, data-link, and network layer services through Altera IP and custom RTL code. We can leverage the lower part of the stack (physical and data-link) to support inter-FPGA communication in our framework. In particular, the Interlaken PHY supported by the protocol stack is especially suited to our needs since it provides many physical and data-link layer services like clock recovery, line coding, framing, and error detection with minimal area overhead.

A key disadvantage of Novo-G#'s stack is its custom nature. Using a standard interface and communication scheme would greatly improve the usability of inter-FPGA communication and encourage its use. Altera's channels extension provides such a standard communication interface. Channels allow OpenCL kernels to communicate with each other through read and write calls, and are recommended by Altera as the best method for inter-kernel data transfers. Unlike the typical OpenCL data transfer model, the host is also completely unaware of the existence of this channel or the data being transferred. For our framework, we have extended the capabilities of AOCL channels, allowing the user to transparently use them for inter-FPGA communication with semantics similar to those of the inter-kernel AOCL channels.

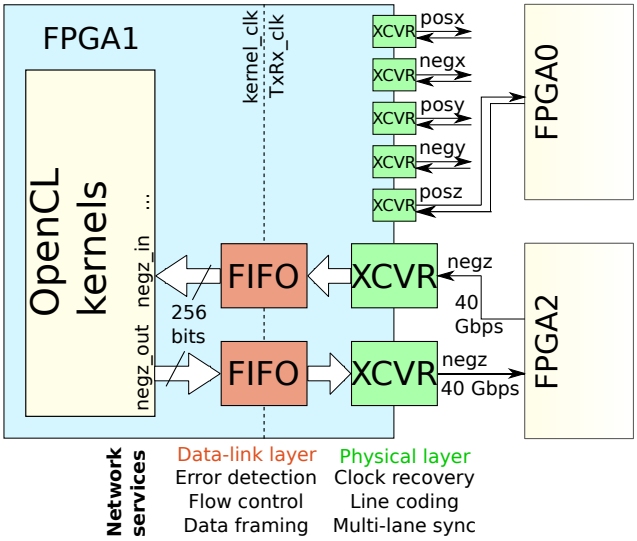


Fig. 1. OpenCL device Architecture depicting inter-FPGA connections. For brevity, not all connections are shown.

B. Integration with framework

We modified the existing ProceV BSP from Gidel to expose the six input and output links to OpenCL as I/O channels with the Altera streaming interface. Each I/O channel can be connected to a single kernel as an input or output channel, and data ordering for these channels is handled in the same manner as the regular channels. Within the BSP, each input/output channel connects to a receive/transmit FIFO respectively, and a controller for the Interlaken PHY. This controller is responsible for generating data frames, control words, and idle words required by the Interlaken protocol. Flow control for the inter-FPGA interface is provided through XOn/XOff bits set during each idle control word. These words are inserted at the start and end of every Interlaken frame, and whenever the transmit FIFO is empty. The FIFO also serves to cross between the (typically) higher clock speed on the kernel interface, to the fixed 200 MHz $TxRx\_clk$ frequency.

The transceiver blocks on the FPGA are configured to use the Interlaken PHY, which provides a number of physical layer services as shown in Figure 1. The IP is configured with a serial rate of 32 Gbps and a parallel interface frequency of 200 MHz, for which a 256-bit data width is sufficient to saturate the channel. While it is possible to run the channels at their rated maximum of 40 Gbps as shown in [6], the transceivers become highly susceptible to parameter variance at that line rate and require the channels to be recalibrated on each FPGA. Such parameter variance makes the task of creating a portable framework extremely difficult since the transceiver parameters are encoded into the base FPGA bitfile.

Once the PHY and interfacing logic was integrated with the rest of the framework, we modified the AOCL platform specification to include the new channel definitions. OpenCL apps using an inter-FPGA channel can identify its source or destination from the channel name (e.g. $posx\_out$). To conform with the AOCL guidelines, we also use an incremental compilation flow using a precompiled "base" revision that requires the end-user to only synthesize the kernel hardware to generate a complete bitfile. The addition of the inter-FPGA communication logic only increases the resource usage of the BSP by 5% of overall FPGA resources and has a negligible impact on the OpenCL kernel frequency.

C. Scaling up

1) Support for multiple devices: The precompiled bitfile generated from the "base" revision is programmed onto all four boards in a server, allowing them to be detected by the AOCL runtime. Support for calling and instantiating kernels on multiple AOCL devices is already built into the runtime, but using the inter-FPGA channels correctly requires each board to be assigned a unique address. For the Novo-G# architecture, a three-dimensional address is assigned based on the bus location and system $hostname$ of each AOCL device. To improve usability for the end user, we provide a pair of C functions to convert between the AOCL device name and the three-dimensional address.

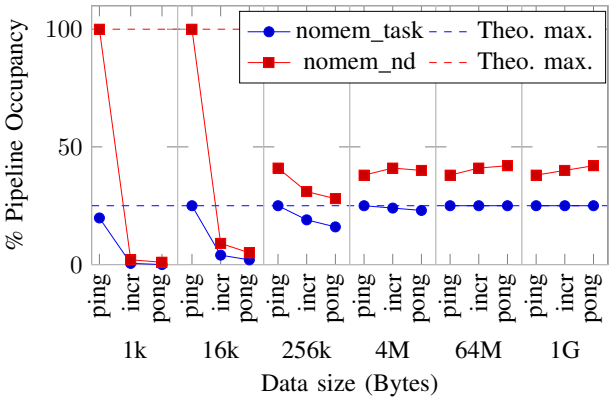
2) Support for multiple nodes: The AOCL runtime environment does not extend beyond a single server. To run applications on multiple servers, we use a basic MPI framework to synchronize and distribute data across the system. When executing on multiple servers, MPI barriers are used to synchronize the FPGAs after they are initialized. However, due to variations in the time required for the transceivers to lock to the incoming data stream, there is a chance that some links may start transmitting data before the transceivers on the other end were ready to receive it. In order to avoid this problem, we preload the bitfile compiled offline on each server, and then create an OpenCL context without reloading the FPGAs. Finally, the above process is not specific to MPI and can potentially be ported to any distributed programming model with which the user is comfortable.

IV. Results

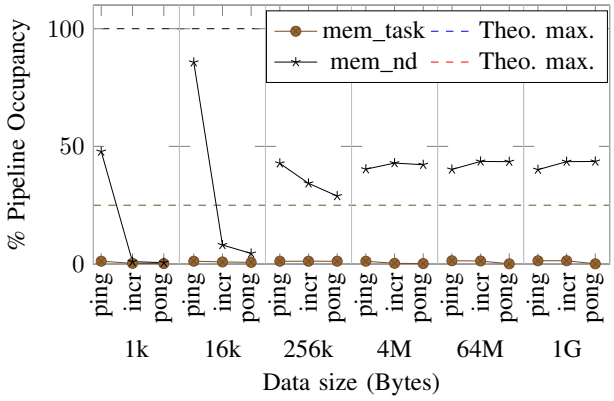
In this section, we describe the three multi-FPGA OpenCL apps that we developed to demonstrate and evaluate the capabilities of multi-FPGA OpenCL. "Ping-pong" is a benchmark run on two FPGAs that enables us to analyze effective channel usage by saturating all the inter-FPGA channels. "Channelizer" showcases how a single-FPGA AOCL app with multiple kernels can be decomposed to run across two FPGAs without loss of performance. "2DFFT" demonstrates how data decomposition can be used to scale AOCL apps effectively onto multiple FPGAs.

A. Ping-pong benchmark

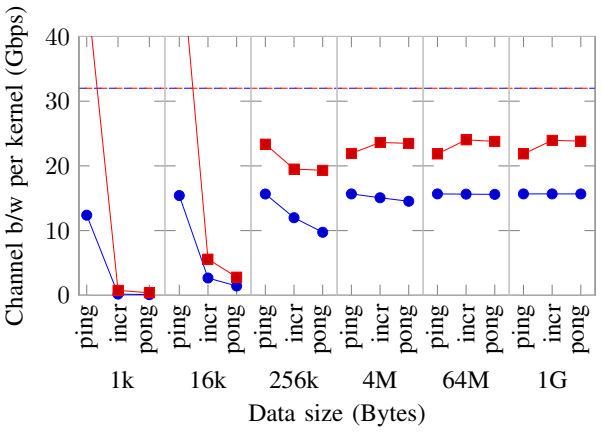
1) Background and Implementation: The ping-pong benchmark is designed to saturate the inter-FPGA channels to the maximum line rate of 32 Gbps per channel, and determine the best configuration to maximize their utilization. The benchmark consists of three kernels instantiated on two FPGAs



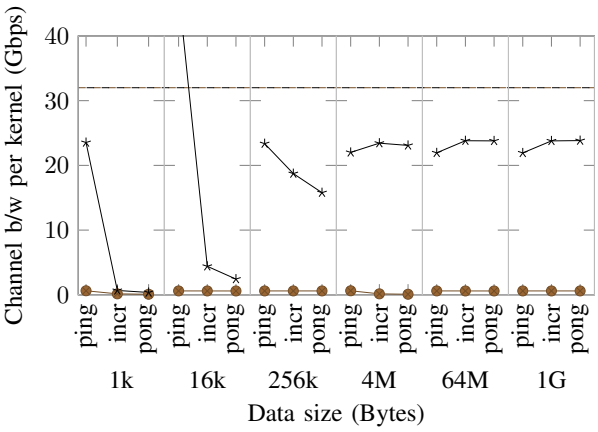
(a) Comparison of pipeline occupancy without memory operations.



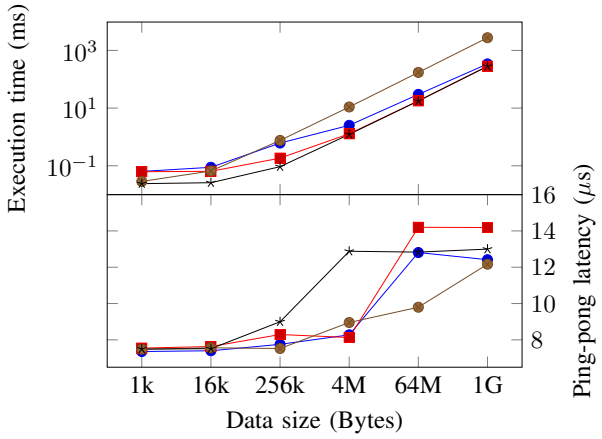
(b) Comparison of pipeline occupancy with memory operations.



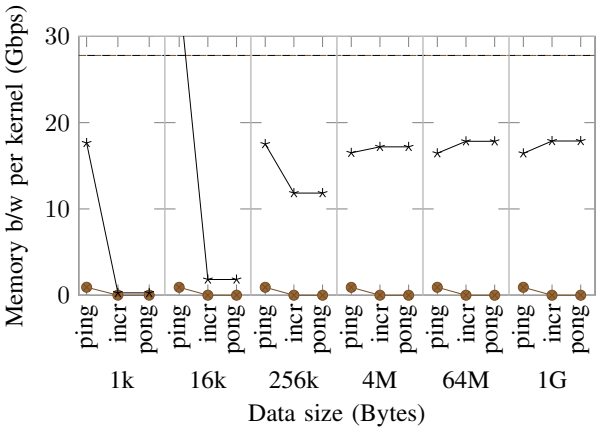
(c) Comparison of inter-FPGA bandwidth without memory operations.



(d) Comparison of inter-FPGA bandwidth with memory operations.



(e) Execution time and latency for all four variants.



(f) Comparison of memory bandwidth with memory operations.

Fig. 2. Collected results from ping-pong OpenCL app variants on Novo-G#.

to saturate the inter-FPGA channels with data. The "ping" kernel on FPGA1 generates data and transmits it over the inter-FPGA link to FPGA2, where it is received by the "incr" (increment) kernel on FPGA2. This kernel performs some simple data manipulation on each word of received data, in this case adding to each word an increment specified by the host and transmitting the data back to FPGA1. The "pong" kernel on FPGA1 receives the modified data and verifies its accuracy. This set of kernels is replicated six times to utilize all inter-FPGA channels.

We created four variants of the above benchmark based on the number of work-items and the source of the input data, to evaluate the efficiency of the inter-FPGA channels in various kernel configurations. First, the three kernels can be implemented using either a single work-item (task) or a multiple work-item (NDRange) scheme. Second, the input of the "ping" kernel can either be generated internally via a counter, or can be read from external memory. In the case of inputs read from memory, the output of the "pong" kernel is also stored in memory so that verification can be done on the host. These four kernel variants ($nomem\_task$, $nomem\_nd$, mem_task, mem_nd) are used to identify bottlenecks in work-item scheduling, pipeline occupancy, memory utilization, and channel bandwidth.

Data on the kernel execution can be gathered using the OpenCL $clGetEventProfilingInfo$ method [20]. For fine-grained details, AOCL also provides support for its own profiling tool [21] that instruments OpenCL code by using extra hardware counters instantiated in the generated pipeline. The AOCL profiler gives useful data in the form of pipeline stalls, occupancy, and channel and memory bandwidth for each channel or memory access instruction in the pipeline.

2) Experiments and Observations: Figure 2 shows data collected from the four variants of the ping-pong benchmark on two ProceV boards in the same server. In each case, we used the AOCL profiler to collect data on the pipeline occupancy, and channel and memory bandwidths, and the $clGetEventProfilingInfo$ method to measure the execution times and end-to-end latency.

Figure 2a compares the pipeline occupancies of the task and NDRange kernels with inputs being internally generated. We see that the task kernels are severely limited in their pipeline utilization and can at most only generate one 256-bit output every four clock cycles. The higher occupancy of the NDRange kernels causes them perform considerably better, since work-items can be scheduled simultaneously to hide the low throughput. While the NDRange kernels could theoretically reach 100% occupancy, they are being limited by the usable channel bandwidth. With inputs being read from memory, as in Figure 2b, the task kernels perform abysmally, limited by their inability to use burst transfers to/from memory, and contention among the six "ping" and six "pong" kernels for the same memory controller. In contrast, the NDRange kernel averages close to the maximum supported burst length of 16 64-bit words shows performance close to the non-memory variant.

```
1  channel ulong4 posx_out __attribute__((depth(0)))
       __attribute__((io("posx_out")));
2  channel ulong4 posx_in __attribute__((depth(0)))
       __attribute__((io("posx_in")));
3  channel ulong4 data_in __attribute__((depth(8)));
4
5  __attribute__((reqd_work_group_size(64,1,1)))
6  kernel void iohelper_nomem() {
7    ulong4 data;
8    data = read_channel_altera(data_in);
9    write_channel_altera(posx_out, data);
10 }
11
12 __attribute__((reqd_work_group_size(64,1,1)))
13 kernel void iohelper_mem(__global ulong4 *
       restrict mem) {
14   uint gid = get_global_id(0);
15   ulong4 data;
16   data = read_channel_altera(posx_in);
17   mem[gid] = data;
18 }
```

Fig. 3. Code listing of NDRange-based I/O channel helper functions, showing channel declaration and use with inter-kernel channels or global memory. Keywords in blue are added by AOCL to support OpenCL channels.

From Figures 2c, 2d, and 2f we can see that the lower pipeline utilization with task kernels severely limits the achievable channel and memory bandwidth, whereas the NDRange kernels achieve an inter-FPGA data rate of approximately 75% of the maximum line rate. It should be noted that for 1kB and 16kB data sizes of the"ping" kernel, the channel and memory bandwidths reported by the profiler are higher than the theoretical maximum; since the channel buffer is large enough to contain the complete data stream, the kernel exits prematurely and artificially inflates the bandwidth reported by the profiler. Finally, Figure 2e compares the overall kernel execution times and the end-to-end latency for a single word for all four variants. In this case, the average latency for NDRange kernels is higher than that of the corresponding task kernels due to the added overhead of scheduling work-items. However, the execution time of the mem_task variant is an order of magnitude worse than the others.

In general, we find that while Altera recommends using task kernels to generate deep pipelines that naturally map well to FPGAs, they are less suited for efficient utilization of inter-FPGA channels or highly contended memory accesses. The use of channels in an AOCL kernel is also restricted to a single call-site within the kernel to allow the AOCL optimizer to improve the pipeline's throughput. However, the scheduling of I/O channel accesses among work-items is well defined by Altera [21]. As long as AOCL's requirements for deterministic work-item ordering are met, the ordered execution behavior of ND-Range kernels proceeds in a sequential order according to the work-item number. By using intra-FPGA channels

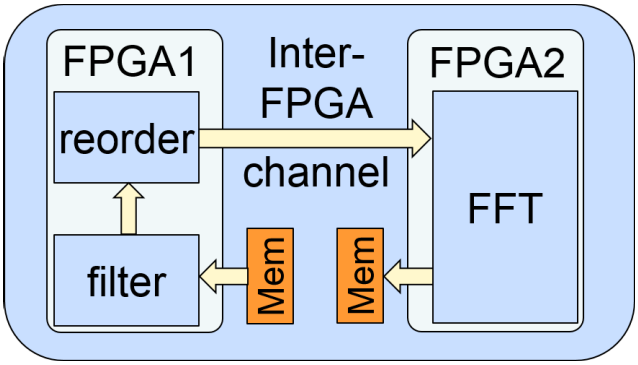


Fig. 4. Channelizer AOCL app task-distributed over two FPGAs.

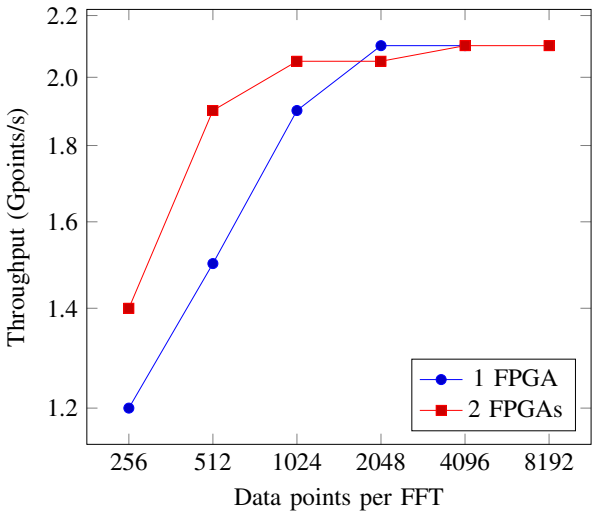


Fig. 5. Results from channelizer OpenCL app on Novo-G#.

to transfer data between task-based kernels and ND-Range kernels, we can be assured of the data ordering within the pipeline. Figure 3 shows two NDRange-based helper functions that can be used to access the I/O channels efficiently. The *iohelper_nomem* kernel shows how output data of an AOCL kernel can be transmitted over the I/O channels, and the *iohelper_mem* kernel shows how data read from an I/O channel can be stored in global memory. Therefore, for the remaining OpenCL apps in this paper, we use task kernels exclusively for computation with uninterrupted (no memory or I/O channel access) deep pipelines, and NDRange kernels to provide efficient data transfer to and from the inter-FPGA channels.

B. Channelizer

1) Background and Implementation: The channelizer [22] OpenCL app is a design example from Altera [23] that implements a polyphase filter followed by a highly optimized 1D FFT [24] on a single FPGA. This example was used by Altera to demonstrate the advantages of using single work-item kernels and channels for inter-kernel communication. The AOCL code defines three task kernels: the polyphase filter, the 1D FFT, and a reorder kernel between the two that serves to rearrange the filter's output data to feed to the FFT. In addition, they use two NDRange kernels to read input filter data from memory and store output FFT data to memory.

We chose this particular example to demonstrate the ease with which existing AOCL code using channels can be modified to work on multiple FPGAs. As shown in Figure 4, the app can be executed on two FPGAs by dividing the AOCL kernels among them and using an inter-FPGA channel to communicate the intermediate data. The I/O helper kernels are not shown in the figure. From the perspective of the code, all that is required is to replace the declaration of one of the inter-kernel channels: $channel\ float8\ REORDER\_TO\_FFT$; with that of an I/O channel: $channel\ float8\ negx\_out\ \_\_attribute\_\_((io("negx\_out")))$; and compile the appropriate kernels to each FPGA. As per the findings from the ping-pong benchmark, we were also able to improve the channel performance by inserting NDRange kernels to supply and receive data from the inter-FPGA channels.

2) Experiments and Observations: We compared the execution of the channelizer app on two FPGAs against the single-FPGA baseline code provided by Altera as shown in Figure 5. The number of input data points supplied to the filter was varied from 256 to 8192, with input vectors being continuously streamed into the hardware to achieve a stable throughput. In general, both implementations are limited by the 2.1 Gpoints/s input rate of the FFT kernel but the greater proportion of setup time with small sample sizes reduces the overall throughput of the app.

A key advantage of using two FPGAs is the greater number of resources available. While the baseline version requires 57% of FPGA resources (72% of DSPs) to implement a 4096-point channelizer, the multi-FPGA version easily doubles the input size supported. An 8192-point channelizer implementation on two FPGAs achieves the same streaming throughput with 30% of FPGA resources (32% of DSPs) on the FPGA with the polyphase filter, and 61% of FPGA resources (75% of DSPs) on the FPGA with the FFT kernel.

C. 2D FFT

1) Background and Implementation: The 2D FFT app is derived from another AOCL design example of the same name [25], which was created to showcase a high-performance radix-4 FFT engine with efficient use of off-chip memory for large problem sizes. We chose this case study since the 2D FFT algorithm scales well, and a distributed implementation would benefit greatly from multidimensional direct communication. The baseline code from Altera is divided into three kernels: an NDRange "fetch" kernel that reads input data from the memory, a task-based "FFT" kernel that performs a 1D FFT on each row of the input data, and an NDRange "transpose" kernel that stores the FFT output back into memory in a transposed fashion. This set of kernels is called twice to perform the complete 2D FFT. To improve the throughout of

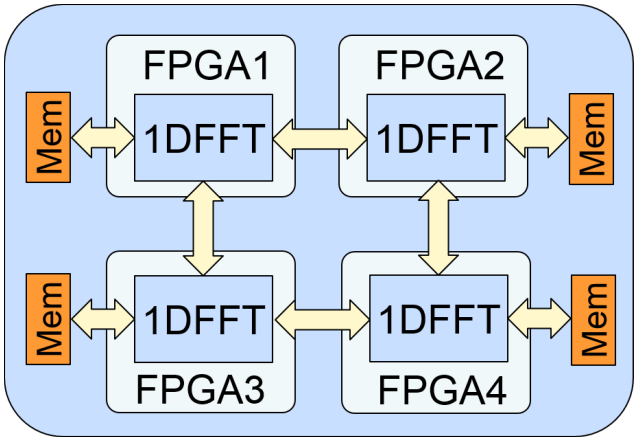


Fig. 6. 2D FFT AOCL app data-distributed over four FPGAs.

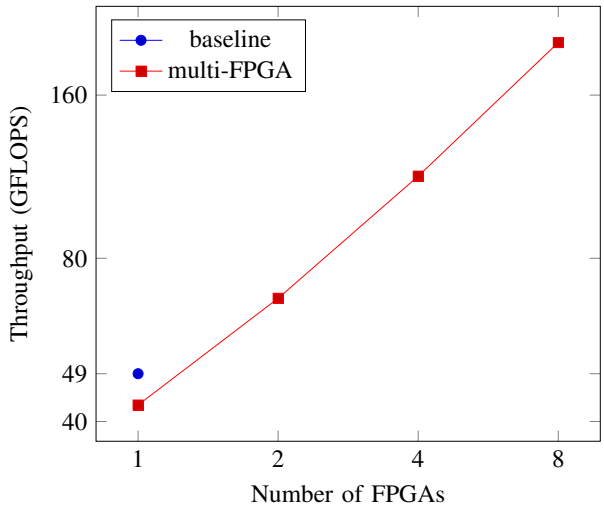


Fig. 7. Results from 2D FFT OpenCL app on Novo-G#.

the app, data transfer between the kernels is done through eight parallel OpenCL channels, with appropriate work-item allocations to the NDRange and task kernels.

The 2D FFT app can be decomposed over a number of FPGAs by using data decomposition as shown in Figure 6. This decomposition requires an exchange of data between each pair of FPGAs after both transpose steps have been performed. We modified the 2D FFT baseline code to use multiple FPGAs by distributing the input matrix over all the FPGAs in question. Each FPGA is now responsible of calculating only a subset of the 1D FFTs in the first stage which required minimal modifications to the "fetch" and "FFT" kernels. The "transform" kernel was modified to reorganize the intermediate data into segments that could be transmitted on the appropriate inter-FPGA channel. We also created an NDRange "receive" kernel to receive data from the neighboring FPGAs and store it in memory. As before, the entire loop is performed twice to obtain the final 2D FFT output in memory, where it can be read out of each FPGA.

2) Experiments and Observations: As shown in Figure 7, we synthesized and executed the multi-FPGA 2D FFT on 1, 2, 4, and 8 FPGAs, for a fixed input size of 1024×1024. The baseline throughput of 49 GFLOPS was achieved by Altera on a BittWare Stratix V D8 board [25]. Our single FPGA implementation performs slightly worse than the baseline due to the difference in clock frequency achieved. As we move up to 8 FPGAs, the speedup increases linearly. One factor behind the strongly linear scaling is the fact that the usable channel bandwidth increases non-linearly with the dimensionality of the network. Beyond 8 FPGAs (a 2×2×2 network), we expect a sub-linear speedup with the number of FPGAs.

V. Conclusion

Novo-G# is a state-of-the-art platform for the acceleration of multi-FPGA communication-intensive apps, but the lengthy and difficult FPGA app development process has limited its use thus far. In this paper, we have presented a multi-FPGA OpenCL framework for Novo-G# geared towards streamlining the end-to-end design flow for multi-FPGA apps. By mapping Novo-G#'s inter-FPGA links to AOCL channels, AOCL kernels on different FPGAs can transparently communicate with each other within the established OpenCL model. We have also created and tested an MPI-based framework to run AOCL programs across multiple servers with multiple nodes each.

Finally, we described the implementation and analysis of three AOCL apps using the multi-FPGA OpenCL framework. The ping-pong benchmark, designed to utilize all the inter-FPGA channels, revealed the ideal manner in which to make use of the channels efficiently, achieving a measured data rate of 24 Gbps on every inter-FPGA channel. The channelizer app showed how easily task-based OpenCL apps could be modified to make use of multiple FPGAs. The 2D FFT app demonstrated how data decomposition of OpenCL apps on multiple FPGAs could be achieved, and showed close to linear speedup on up to 8 FPGAs against an optimized baseline.

The multi-FPGA OpenCL framework described here was designed with the Novo-G# system in mind, but it can easily be ported to other AOCL platforms that provides support for inter-FPGA communication. The framework employs technologies readily available from Altera and does not depend on a particular network topology, connectivity, or protocol; these can be readily customized to support the framework on a new platform. Further, the lessons learned from our evaluation of task-based and NDRange-based kernels are dependent only on the AOCL compiler, and are thus broadly applicable to any AOCL platform that makes use of channels.

There are a number of improvements that can be made to this framework in the future. Adding the ability to tune the transceiver parameters at runtime could potentially allow the inter-FPGA channels to operate closer to the 40 Gbps line rate. Additionally, integrating the network layer router from [6] would allow for transparent packet routing within the 3D torus network. Both of these improvements however, are unlikely to fit within the established AOCL model or the OpenCL 2.0

specification and would therefore require significant changes to the AOCL hardware and runtime to support them.

The recent use of FPGAs in data centers has opened up an avenue for more large-scale RC system installations. However, there are many programmability and usability issues that must be addressed if such systems are to find greater acceptance in the HPC community. With our multi-FPGA OpenCL framework, we have tried to address the high barrier-to-entry of RTL design through the use of HLS, improve the state of distributed RC app design tools and present a communication model based on standard OpenCL constructs to simplify the creation of multi-FPGA designs.

ACKNOWLEDGMENT

This work was partially supported by CHREC members and the I/UCRC Program of the National Science Foundation under Grant No. IIP-1161022, and by the division of Computer and Network Systems under Grant No. CNS-1405790. We gratefully acknowledge tools and devices provided by Altera. We especially acknowledge the extended hardware support provided by Gidel.

VI. REFERENCES

REFERENCES

[1] A. Putnam, A. M. Caulfield, E. S. Chung, D. Chiou, K. Constantinides, J. Demme, H. Esmaeilzadeh, J. Fowers, G. Prashanth, G. Jan, G. Michael, H. S. Hauck, S. Heil, A. Hormati, J.-Y. Kim, S. Lanka, J. Larus, E. Peterson, S. Pope, A. Smith, J. Thong, P. Yi, and X. D. Burger, "A reconfigurable fabric for accelerating large-scale datacenter services," in *Proceedings of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 13–24. [Online]. Available: http://dl.acm.org/citation.cfm?id=2665671.2665678

[2] "Altera and Baidu Collaborate on FPGA-based Acceleration for Cloud Data Centers — Press Release," September 2014, retrieved on: 2015-02-25. [Online]. Available: http://newsroom.altera.com/press-releases/altera-baidu-fpga-cloud-data-centers.htm

[3] "Accelerating open innovation for emerging cloud workloads — intel," 2016. [Online]. Available: http://itpeernetwork.intel.com/accelerating-open-innovation-for-emerging-cloud-workloads/

[4] A. D. George, H. Lam, A. Lawande, C. Pascoe, and G. Stitt, "Novo-g: A view at the hpc crossroads for scientific computing." in *ERSA*, 2010, pp. 21–30.

[5] A. George, H. Lam, and G. Stitt, "Novo-G: At the Forefront of Scalable Reconfigurable Supercomputing," *Computing in Science & Engineering*, vol. 13, no. 1, pp. 82–86, Jan. 2011. [Online]. Available: http://scitation.aip.org/content/aip/journal/cise/13/1/10.1109/MCSE.2011.11

[6] A. G. Lawande, A. D. George, and H. Lam, "Novo-g#: a multidimensional torus-based reconfigurable cluster for molecular dynamics," *Concurrency and Computation: Practice and Experience*, vol. 28, no. 8, pp. 2374–2393, 2016, cpe.3565. [Online]. Available: http://dx.doi.org/10.1002/cpe.3565

[7] "Gidel Products — Gidel Ltd." retrieved on: 2015-02-25. [Online]. Available: http://www.gidel.com/Products.htm

[8] "Interlaken protocol product page — altera," 2016. [Online]. Available: https://www.altera.com/products/intellectual-property/ip/interface-protocols/m-alt-interlaken.html

[9] O. Segal, N. Nasiri, M. Margala, and W. Vanderbauwhede, "High level programming of fpgas for hpc and data centric applications," in *High Performance Extreme Computing Conference (HPEC), 2014 IEEE*, Sept 2014, pp. 1–3.

[10] S. Tatsumi, M. Hariyama, M. Miura, K. Ito, and T. Aoki, "Opencl-based design of an fpga accelerator for phase-based correspondence matching," in *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA)*, 2015, p. 90.

[11] H. Giefers, R. Polig, and C. Hagleitner, "Accelerating arithmetic kernels with coherent attached fpga coprocessors," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 1072–1077.

[12] D. F. Bacon, R. Rabbah, and S. Shukla, "Fpga programming for the masses," *Commun. ACM*, vol. 56, no. 4, pp. 56–63, Apr. 2013. [Online]. Available: http://doi.acm.org/10.1145/2436256.2436271

[13] Z. Zhang, Y. Fan, W. Jiang, G. Han, C. Yang, and J. Cong, *AutoPilot: A Platform-Based ESL Synthesis System*. Dordrecht: Springer Netherlands, 2008, pp. 99–112. [Online]. Available: http://dx.doi.org/10.1007/978-1-4020-8588-8_6

[14] "Catapult high-level synthesis - mentor graphics," 2016. [Online]. Available: https://www.mentor.com/hls-lp/catapult-high-level-synthesis/

[15] "Impulse codeveloper c-to-fpga tools — impulse accelerated," 2016. [Online]. Available: http://www.impulseaccelerated.com/products_universal.htm

[16] M. Lin, I. Lebedev, and J. Wawrzynek, "Openrcl: Low-power high-performance computing with reconfigurable devices," in *2010 International Conference on Field Programmable Logic and Applications*, Aug 2010, pp. 458–463.

[17] M. Owaida, N. Bellas, K. Daloukas, and C. D. Antonopoulos, "Synthesis of platform architectures from opencl programs," in *Field-Programmable Custom Computing Machines (FCCM), 2011 IEEE 19th Annual International Symposium on*, May 2011, pp. 186–193.

[18] P. O. Jskelinen, C. S. de La Lama, P. Huerta, and J. H. Takala, "Opencl-based design methodology for application-specific processors," in *Embedded Computer Systems (SAMOS), 2010 International Conference on*, July 2010, pp. 223–230.

[19] A. Papakonstantinou, K. Gururaj, J. A. Stratton, D. Chen, J. Cong, and W. M. W. Hwu, "Fcuda: Enabling efficient compilation of cuda kernels onto fpgas," in *Application Specific Processors, 2009. SASP '09. IEEE 7th Symposium on*, July 2009, pp. 35–42.

[20] "clgeteventprofilinginfo command manual — khronos," 2016. [Online]. Available: https://www.khronos.org/registry/cl/sdk/1.0/docs/man/xhtml/clGetEventProfilingInfo.html

[21] Altera Inc., 2016. [Online]. Available: https://www.altera.com/en_US/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf

[22] J. Freeman, *FPGA Channelizer Design in OpenCL*, 1st ed. Altera Toronto Technology Center, 2016. [Online]. Available: https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/support/examples/download/exm_opencl_channelizer.pdf

[23] "Channelizer design example — altera," 2016. [Online]. Available: https://www.altera.com/support/support-resources/design-examples/design-software/opencl/channelizer.html

[24] M. Garrido, J. Grajal, M. A. Sanchez, and O. Gustafsson, "Pipelined radix- 2^k feedforward fft architectures," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 1, pp. 23–32, Jan 2013.

[25] "Opencl 2d fast fourier transform design example — altera," 2016. [Online]. Available: https://www.altera.com/support/support-resources/design-examples/design-software/opencl/fft-2d.html