Exploring Portability and Performance of OpenCL FPGA Kernels on Intel HARPv2

Anthony M. Cabrera
Roger D. Chamberlain

Anthony M. Cabrera and Roger D. Chamberlain, “Exploring Portability and Performance of OpenCL FPGA Kernels on Intel HARPv2,” in *Proc. of 7th International Workshop on OpenCL* (IWOCL), May 2019. DOI: 10.1145/3318170.3318180

Dept. of Computer Science and Engineering
McKelvey School of Engineering
Washington University in St. Louis

Exploring Portability and Performance of OpenCL FPGA Kernels on Intel HARPv2

Anthony M. Cabrera
Washington University in St. Louis
St. Louis, Missouri, USA
acabrera@wustl.edu

Roger D. Chamberlain
Washington University in St. Louis
St. Louis, Missouri, USA
roger@wustl.edu

ABSTRACT

FPGAs offer a heterogenous compute solution to the continuous desire for increased performance by enabling the creation of application-specific hardware that accelerates computation. While the barrier to entry has historically been steep, advances in High Level Synthesis (HLS) are making FPGAs more accessible. Specifically, the Intel FPGA OpenCL SDK allows software designers to abstract away low level details of architecting hardware on an FPGA and allows them to author computational kernels in a higher level language. Furthermore, Intel has developed a system that incorporates both a multicore Xeon CPU and Arria 10 FPGA into the same chip package as part of the Heterogeneous Accelerator Research Program (HARP) that can be targeted by their SDK.

In this work, we target the second iteration of the HARP platform (HARPv2) using HLS through porting of OpenCL kernels originally written for FPGAs connected via a PCIe bus. We evaluate the HARPv2 system's performance against previously reported results, explore the portability of kernels through a hardware design space search, and empirically show the benefits of using the shared virtual memory (SVM) abstraction over explicit reads and writes.

CCS CONCEPTS

• **Computer systems organization → Reconfigurable computing**; **Heterogeneous (hybrid) systems**.

KEYWORDS

High Level Synthesis, Needleman-Wunsch, FPGA, Design Space Search, Shared Virtual Memory

ACM Reference Format:
Anthony M. Cabrera and Roger D. Chamberlain. 2019. Exploring Portability and Performance of OpenCL FPGA Kernels on Intel HARPv2. In *International Workshop on OpenCL (IWOCL'19), May 13–15, 2019, Boston, MA, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3318170.3318180

1 INTRODUCTION

As the end of Moore's law draws nearer, researchers across disciplines are looking beyond relying on performance increases through packing more transistors into CPUs and scaling CPU clock frequencies. Outside of multicore CPUs, people are turning towards heterogeneous computing solutions that incorporate hardware co-processors such as Graphics Processing Units (GPUs) and Field Progammable Gate Arrays (FPGAs) to accelerate computation. The former has become ubiquitous in desktop, server, and cloud environments and has an established and mature ecosystem.

The widespread use of FPGAs, however, is still nascent while their presence is burgeoning. This forward progress is reflected in industry with companies like Amazon and Microsoft equipping their data center nodes with FPGAs [2, 7, 21] and Intel acquiring FPGA manufacturer Altera. Additionally, there is a growing research trend toward harnessing the reconfigurability of FPGAs towards salient applications like accelerating neural networks [5, 11, 30], biocomputation [15, 18, 19], and many other applications [17, 24, 26, 29, 34].

A common way to incorporate hardware acclerators like GPUs and FPGAs into a computer system is to attach them through a PCIe bus, which keeps hardware costs relatively low. In spite of this, the use of FPGAs has not experienced the widespread adoption that GPUs have seen, in part because of all the difficulties inherent in their use. Historically, FPGA developers have needed to be well versed in electronic circuits and digital logic design. This includes knowledge of low-level hardware interaction at the register-transfer level (RTL) and handling timing constraints at a clock cycle granularity, as well as domain-specific knowledge of computer-aided design tools and workflows specific to FPGA design and development. This is generally outside of the skillset of most software developers.

One of the steepest barriers to using FPGAs is expressing a design in the first place using traditional hardware description languages (HDLs) like VHDL and Verilog, which requires the domain specific knowledge previously mentioned. A current research direction in lowering the barrier is High Level Synthesis (HLS), which allows a programmer to express a kernel of computation in a higher level language like C or C++ for deployment onto an FPGA. This circumvents the problem of having to learn an HDL to express a kernel and its low level interfaces, reduces the amount that a programmer has to understand about FPGA microarchitecture, and abstracts away the lower level details of using FPGAs.

One way that companies that build PCIe cards around FPGAs from Intel and Xilinx enable the use of HLS is through making their solutions OpenCL compliant. This involves providing a Board Support Package (BSP) that provides the interface between host and device, as well as parameters that are used by an offline compiler to synthesize, place, and route a design onto whatever FPGA is used on the card. In addition to PCIe cards, Intel has also developed a system that incorporates both a multicore Xeon CPU and Arria 10

IWOCL'19, May 13–15, 2019, Boston, MA, USA

ACM ISBN 978-1-4503-6230-6/19/05.
https://doi.org/10.1145/3318170.3318180

FPGA into the same chip package (described more in Section 2.2). This particular project is known as the Heterogeneous Accelerator Research Platform, or HARP. In addition to being able to author designs using an HDL, Intel has provided the infrastructure to use the Intel FPGA OpenCL SDK for FPGA development. While there have been recent publications targeting this system with a traditional FPGA design flow [1, 9, 24, 25, 28, 31], not much is known about the experience, feasibility, and performance of targeting a HARP system using OpenCL.

In this paper we will target the second iteration of the HARP CPU+FPGA processor (HARPv2) through HLS using the Intel OpenCL SDK for FPGA to explore the portability and performance of OpenCL FPGA kernels. Specifically, we use OpenCL kernels authored for an FPGA attached via PCIe card that perform the Needleman-Wunsch algorithm [33] and port them to the HARPv2 system. Then, we evaluate and compare our results to ones previously reported, present our findings of portabilty through exploring the hardware design space, and show the benefit of using the Shared Virtual Memory (SVM) abstraction implemented for the HARP system.

The rest of the paper is as follows. Section 2 details preliminary information about FPGAs, the HARP system, and best practices for authoring kernels that target FPGAs through the Intel OpenCL SDK for FPGAs. Section 3 describes the related work of HLS utilization and development as well as hardware designs that target the HARP system. Section 4 outlines each of the different kernel versions used for the target application, the experience of porting the kernels to work on our test HARP system, describing the hardware design space, and enabling the use of the SVM abstraction. Finally, Section 5 presents the results from comparing our results to those in literature, the results from a hardware design space exploration, and the results of using the SVM abstraction, and Section 6 concludes and presents directions for future work.

2 PRELIMINARIES

2.1 FPGA

Field Progammable Gate Arrays (FPGAs) are integrated circuits that include programmable logic blocks, hardened logic blocks such as Digital Signal Processors (DSPs) and floating point units (FPUs), block RAMs (BRAMs, and referred to as M20K blocks for Intel FPGAs), and reconfigurable routing circuitry to connect these components together and to hardened I/O logic in order to interface with external hardware. FPGAs tend to occupy the middle ground between general purpose CPUs and application specific integrated circuits (ASICs) in terms of programmabilty, performance, and power dissipation. FPGA developers traditionally design hardware using Hardware Description Languages (HDLs) such as VHDL or Verilog. This allows them to tailor hardware to a specific application. This usually results in better performance than CPUs. The effective use of hardware that is specific only to the problem also leads to lower power dissipation.

2.2 Intel HARPv2

The second iteration of the Heterogeneous Architecture Research Platform (HARPv2) system incorporates a 14 core Intel Broadwell Xeon CPU with an Intel Arria 10 GX1150 in the same chip package, where both the CPU and FPGA share the same memory. Relative to the Stratix V GX A7 in HARPv1, the FPGA in HARPv2 has 1.06 times more M20K blocks, 1.82 times more logic blocks and registers, 5.93 times more DSP blocks, and is located on the same chip package as opposed to a different socket. Integrating the CPU and FPGA on the same package is different from traditional FPGA accelerator solutions that are connected via PCIe slot or on their own development board. The FPGA is connected to the CPU through three physical channels: one through Intel's QuickPath Interconnect (QPI), and the other two through PCIe lanes. Intel also provides the low level interface hardware for the FPGA through their Board Support Package (BSP). Faict presents an excellent overview of the HARP system in [10].

2.3 Intel FPGA OpenCL SDK

While it is possible to target the HARPv2 system by writing custom HDL, it is also possible to use HLS using the Intel FPGA OpenCL SDK. This SDK is an implementation of the OpenCL standard API that allows for programmers to author both host and device code in a high level language. The SDK provides a runtime environment (RTE) that controls the execution of kernels on the FPGA. All of the low level interfaces and drivers that facilitate the interaction between the host and target device(s) are included in the BSP, traditionally provided by the board manufacturer. In the case of the HARPv2 platform, a pilot BSP is provided by Intel. The Intel OpenCL FPGA SDK provides an offline compiler that takes an OpenCL kernel, creates an HDL representation of that design in Verilog, synthesizes that into logical FPGA elements (RTL), maps that design into FPGA components (e.g. logic blocks, I/O blocks), places the mapped design onto the target FPGA, and routes the design.

2.3.1 NDRange vs. Single Work Item Programming Model. There are two programming models that can be used to author OpenCL kernels that target FPGAs: the NDRRange (NDR) and Single Work-Item (SWI) models. These models are pictorially described in Figure 1. The NDRange (NDR) model expresses kernels through specifying a global amount of work (i.e. global work size) to do in (up to) a 3-dimensional space, and a local amount of work to do (i.e. local work size) in that same space that will be scheduled for execution on a processing element. In Figure 1, the NDRange kernel is specified in a 1-dimensional space. Kernel execution, then, must be enqueued from the host side to make sure all global work items will be executed. Each work item is then scheduled by a hardware scheduler on the FPGA side. The Single Work Item (SWI) model expresses kernels by setting the global and local work size to 1 in all dimensions so that all computation is handled by a single work item. In both cases, a custom pipeline is created for computation, as shown in Figure 1.

Intel recommends using the SWI model if the target kernel contains many loop and memory dependencies [14]. This allows the offline compiler to have a global view of all computation so it can account for dependencies when constructing a custom pipeline. Ideally, iterations can then be launched every clock cycle. Additionally, fine-grained sharing between loop iterations in an NDR kernel requires an intricate mechanism that involves local memory

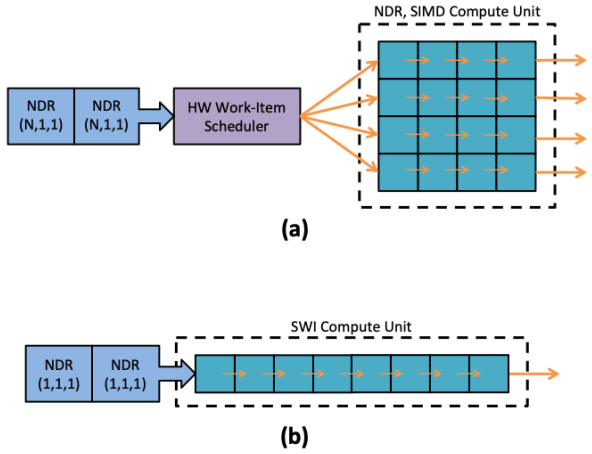

Figure 1: (a) The NDRange model relies on using multiple work items to perform kernel computations. Each of these work items must be scheduled for execution onto the compute unit by a hardware scheduler implemented on the FPGA. In this case, there are two instances of some 1D NDRange kernel enqueued for execution that each have `N` global work items that need to be executed. The pipelined compute unit in this case has been vectorized by a factor of 4. (b) The Single Work Item kernel uses only one work item and thus does not need a hardware scheduler. Single work items rely on pipelining to exploit instruction-level parallelism and resolving of loop and memory dependencies between iterations without the use of costly memory barriers.

and barriers, and this leads to suboptimal kernel performance. Zohouri [32] takes this further and says that NDRange kernels should only be employed if loops cannot be fully pipelined due to variable exit conditions, complex loop-carried dependencies, or random external memory accesses. For all other cases, the SWI model should be employed.

2.4 Needleman-Wunsch

The workload targeted in this paper is the Needleman-Wunsch algorithm. It is a dynamic programming algorithm used for globally aligning a pair of protein or nucleotide sequences. The end result of the algorithm is a substitution matrix that is used to trace back the optimal global alignment of the two sequences. The general structure of the implementation (without any notion of blocking or parallelism) is outlined in Algorithm 1.

The size of the substitution matrix is determined by the length of the two strings to be compared. In this case, the two strings are both of size N, and an extra row and column are added for initial conditions. The substitution matrix is populated by iterating across all elements in each row as detailed in the nested `for` loop starting in line 6. Each element $subst\_matrix[i][j]$ is calculated as a function of the elements to the left, top, and top left from the current element, as well as a similarity score from $score\_matrix[i][j]$ and a predetermined penalty constant for alignment gaps.

Algorithm 1: Needleman-Wunsch Algorithm

```
1:  new int subst_matrix[N+1][N+1], score_matrix[N+1][N+1]
2:  new int gap_penalty
3:  initialize first row and column of subst_matrix
4:  initialize score_matrix
5:  initialize gap_penalty
6:  for i ← 1 to N + 1 do
7:    for j ← 1 to N + 1 do
8:      top = subst_matrix[i − 1][j] - gap_penalty
9:      left = subst_matrix[i][j − 1] - gap_penalty
10:     top_left = subst_matrix[i − 1][j − 1] + score_matrix[i][j]
11:     subst_matrix[i][j] = max(top, left, top_left)
12:   end for
13: end for
```

The Needleman-Wunsch algorithm is included as part of the Rodinia benchmarking suite [8] developed by Che et al. In Rodinia, the Needleman-Wunsch algorithm has 3 different implementations: an OpenMP implementation that targets multicore CPUs, a CUDA implementation that targets NVIDIA GPUs, and an OpenCL implementation for any accelerator that is compliant with the OpenCL standard. Zohouri et al. extended this work by authoring kernels using the Intel FPGA OpenCL SDK to target FPGAs connected as a card on the PCIe bus [32, 33]. They do this for a subset of the Rodinia suite and show the advantages and disadvantages of FPGAs as accelerators compared to GPUs and multicore CPUs. We use the Needleman-Wunsch FPGA kernels from Zohouri et al. for experimentation in this work, and detail each version in Section 4.1.

3 RELATED WORK

3.1 HARP for Acceleration

Since its inception, there have been many projects that have demonstrated the benefits of using the HARP system in a variety of different applications and domains. Podili et al. use the HARP as their experimental system in using Winograd FFTs to speed up convolutional layers in Convolutional Neural Networks [20]. Alves et al. utilize the low-latency QPI channel for collision detection algorithms to demonstrate the HARP's feasibility for real-time applications [1]. Sidler et al. exploit the shared memory feature of the HARP system to reduce superfluous data movement in pattern matching for databases [24]. Stitt et al. develop a scalable window generator architecture for sliding window applications, which are a common pattern in FPGA design, to take advantage of the increased memory bandwidth in the HARPv2 system and reported future memory bandwidth increases for FPGAs [25]. Wang leverages the tighter coupling of CPU and FPGA in the HARP system as a heterogeneous platform for accelerating graph processing [28]. In all of these cases, though, custom RTL is written to express the hardware and low-level interfaces for the HARP system, which is a skill not generally in the toolbox of the modern software developer. In this work, we explore the performance and portability of OpenCL kernels, which allow developers to describe accelerator functions in a higher level language.

3.2 OpenCL + FPGA

There have been a number of recent case studies that target FPGAs using OpenCL. Jin and Finkel evaluate the performance of varying the number of replicated compute units for an OpenCL kernel that computes an MD5 hash [16]. Sanaullah and Herbordt use the Verilog created by the Intel FPGA OpenCL SDK offline compiler for an OpenCL kernel that describe a fast fourier transform, apply code structure optimizations, and outperform vendor IP-based designs while also being able to fit this modified design into existing FPGA solutions that use FFTs [22]. The kernels used in this paper are sourced from work done by Zohouri et al. that aims to evalute and optimize OpenCL kernels taken from the Rodinia suite [8] to evaluate the effectiveness of FPGAs in high performance computing applications [32, 33].

In all of these works, the OpenCL kernels are authored for FPGAs attached via a PCIe bus. This work specifically evaluates the portability and performance of OpenCL FPGA system on the HARPv2 system, in which the CPU and FPGA are located on the same chip package and share a common memory. To date, the only other instance of using OpenCL to target the HARPv2 system is Faict's exploration of OpenCL for guided image filtering [10]. In this work, we present a more exhaustive exploration of the hardware design space.

3.3 High Level Synthesis and Design

In addition to the Intel FPGA OpenCL SDK, there are other ways to leverage High Level Synthesis (HLS) and design to target FPGAs. One of the earliest HLS languages that preceeded OpenCL for FPGAs was the Streams-C language and compiler, implemented by Gokhale et al. [12], that allowed programmers to author streaming kernels in a C-based language. They also quantified the tradeoffs between performance and ease of programmability using HLS. LegUp, developed by Canis et al., takes a C program as input and automates the process of finding segments of code that can be accelerated on an FPGA [6]. Bachrach et al. develop a hardware construction language called Chisel in the Scala programming language in order to design hardware using object-oriented principles and functional programming [3]. It is important to make the distinction that Chisel is not a "C-to-Gates" form of HLS; this solution allows for a more expressive description of hardware using higher level ideas like object-oriented programming.

4 METHODS

This section describes the kernels built for and deployed on the Intel HARPv2 system. We use the same kernel version enumeration from [32]. The kernels are built using the offline compiler provided in the Intel FPGA OpenCL SDK and a custom release of the 16.0 version of the Intel Quartus Prime tool suite that accomodates the HARPv2 system. This is the most recent version of the Quartus tools that is supported by the test system. Minimal changes were made to the original host source code during the porting process. The only change made to the kernel code was correcting an indexing error in Kernel Version 5 that left the last column and the last two rows unprocessed. The process of exploring the hardware design space is detailed next. Finally, we detail how we enable the HARPv2 system, including using the Shared Virtual Memory (SVM) abstraction.

4.1 Description of Each Kernel Version

The kernel versions used in this paper are from [8, 32, 33] and are described in the following subsections. Versions 0 and 2 are designed using the NDR paradigm, and Versions 1, 3, and 5 use the SWI paradigm. Versions 2 and 3 apply basic level optimizations to their preceding versions, and Version 5 implements a new design using the SWI model. Each kernel was built for and deployed on the Intel HARPv2 system for comparison to the prior work that evaluates performance with FPGAs that are connected via PCIe card.

4.1.1 Version 0. This kernel takes the OpenCL implementation from [8], which uses 2D blocking to subdivide the problem with no modifications and is used as the performance baseline. Its implementation follows the NDR paradigm. The kernel is divided into two separate kernel functions that perform the same computation but are indexed differently to compute the upper and lower trangular, respectively, of a given 2D block. Each function takes advantage of diagonal parallelism in two ways: thread- and block-level parallelism. Once an element or block of index (i, j) is computed, it satisfies the dependencies for the $(i, j+1)^{th}$ and $(i+1, j)^{th}$ elements or blocks, and allows them to be computed in parallel. The size of the 2D block is determined by a user-defined variable named `BSIZE`.

4.1.2 Version 1. This kernel uses a doubly nested `for` loop as outlined in Algorithm 1 and takes no steps to guide the synthesis tools on how to better achieve computational parallelism in the resulting custom pipeline.

4.1.3 Version 2. This kernel applies basic compiler-level optimizations [33] to Version 0 in two ways. The first is through setting the maximum work group size. This constraint allows the compiler to perform more aggressive optimizations without wasting precious hardware resources [14]. Additionally, setting the size also enables the second optimization: kernel vectorization. This is achieved by adding the `SIMD` attribute to the kernels in order to vectorize them. This allows work items to execute more data. In this work, the kernels are vectorized by a factor of 2.

4.1.4 Version 3. Version 3 improves on Version 1 in two ways. The first is by adding a register to cache the result of the current element so that it can satisfy the left dependency for the next iteration and avoid an external memory access. The second is by adding the compiler pragma `ivdep` on the substitution matrix to prevent compiler from assuming false load/store dependencies on that global buffer (since the current element being computed depends on previously computed values in that buffer) and to decrease stall cycles per loop iteration.

4.1.5 Version 5. Version 5 is a kernel programmed using the SWI model. Instead of iterating across elements left to right as in previous SWI implementations, Version 5 takes advantage of diagonal parallelism similar to the NDR kernel versions. It divides the substitution matrix into groups of rows, i.e. 1D blocks. The number of rows in each 1D block is set by a tunable hardware parameter named `BSIZE`. Each 1D block of the matrix is processed by dividing the block into column chunks. Specifically, there is a hardware parameter named `PAR` that sets how many columns are in each chunk.

The chunks of columns are processed in a diagonal fashion, and wrap around to the next chunk of columns once the current one is finished. Since all loops are successfully fully unrolled, there is no need to employ any kernel vectorization. The kernel is done processing once all of the columns of the 1D chunk have been computed. The exit condition is precomputed on the host side.

While there are other optimizations employed as described in [32], the main optimization is use of shift registers as local storage to satisfy dependencies. This is done in two ways. The first is by using shift registers to hold onto computed elements of the substitution matrix between iterations of the loop. This is similar to the idea of caching a computed element in Version 3, but the shift registers act as buffers that satisfy dependencies across multiple rows and columns instead of just the next element to be computed in the substitution matrix. The size of these shift registers are a function of `BSIZE` and `PAR`. The second way is by creating 2D shift registers and utilizing them in a staircase fashion as shown in Figure 2. Because each 1D block is traversed in a diagonal fashion, the access patterns of global memory are not spatially local. To this end, the staircase shift registers are employed such that reads and writes to global memory can still be coalesced, but are buffered until they are needed to compute an element in the substitution matrix.

4.2 Hardware Design Space Search

In [32], Zohouri reports the optimal parameter settings for `BSIZE` in kernel Versions 0 and 2 and the optimal settings for `BSIZE` and `PAR` for Version 5 for the PCIe-connected Stratix V and Arria 10 FPGAs that he uses in his experimentation. These parameters are effectively hardware design knobs that are exposed by the kernel designer. `BSIZE` controls how much of the substitution matrix was computed for the NDR kernels, and is a parameter for sizing some shift registers in kernel Version 5. The `PAR` parameter controls the degree of parallelism for kernel Version 5, i.e. how many substitution matrix elements can be processed at the end of a loop iteration, and determines how large to make the staircase shift register array, as shown in Figure 2. In order to find the parameter configurations for the Intel HARPv2 system that produce optimal performance, we define a hardware design space by creating a range of values that `BSIZE` can take for Versions 0 and 2, and a range of values that `BSIZE` and `PAR` can take for kernel Version 5. This range is defined, in part, by what configurations the tools are able to successfully build.

4.3 Shared Virtual Memory

Though the Intel HARPv2 nodes used in this paper are technically only OpenCL 1.0 compliant (as reported by querying the `CL_DEVICE_VERSION` parameter of the device), they do support the feature of using Shared Virtual Memory (SVM) implemented as an extension to the OpenCL 1.0 API. It is worth noting, though, that devices compliant with versions of OpenCL 2.0 and up are required to support SVM.

Instead of having to explicitly enqueue writes and reads to and from the HARPv2 FPGA, shared memory is allocated on the host side and then is pointed to as a special SVM kernel parameter from the host code. In order to utilize this feature in the HARPv2 system, the host code needed to be edited in the following ways: all previously created `cl_mem` objects created and freed for the device were removed and replaced with shared memory allocated by `clSVMMallocAltera()` and `clSVMFreeAltera()`, respectively. Enqueueing writes and reads to `cl_mem` objects were removed. Finally, calls to `clSetKernelArg()` that pointed to `cl_mem` objects were replaced with `setArgSVMAltera()`. Conveniently, no changes need to be made to the kernel code to accomdate using SVM instead of explicit reads and writes. Thus, kernels do not need to be rebuilt to accomodate using the SVM feature.

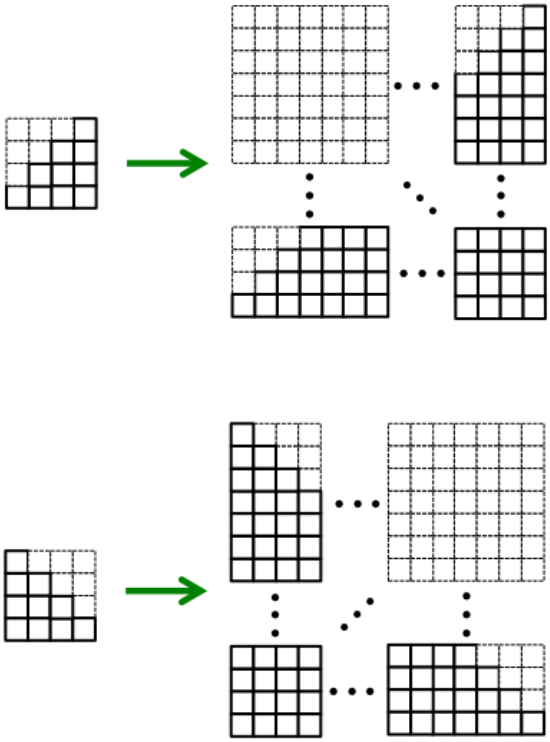

Figure 2: Hardware effect of staircase shift register when sweeping the `PAR` parameter. A 2D array of shift registers is allocated in the OpenCL kernel, but only half of the structure is used. This is represented by boldening the lines of the utilized shift registers and using dashed lines to represent the shift registers that are unused. The top figure is the shape of the arrays used to buffer data and coalesce reads from global memory, and the bottom is the shape used to buffer data and coalesce writes to global memory.

5 RESULTS AND DISCUSSION

5.1 FPGA Kernel Results

Table 1 shows the results of building and evaluating each kernel version described in Section 4.1 on the Intel HARPv2 system and compares them to previously reported results in [32]. Specifically, any row that contains "HARP" in the "FPGA" column contains our results for the Intel HARPv2 platform, and all other data is from [32]. The percentages reported are how much of that particular

Version	Opt. Level	Kernel Type	FPGA	Time (sec)	f_{max} (MHz)	Logic	M20K Bits	M20K Blocks	DSP	Speedup
v0	None	NDR	Stratix V, PCie	9.937	267.52	27%	16%	30%	6%	1.00
			Arria 10, HARP	13.367	211.77	25%	39%	25%	1%	0.74
v1	None	SWI	Stratix V, PCie	203.864	304.50	20%	5%	17%	< 1%	0.05
			Arria 10, HARP	830.131	256.6	26%	9%	18%	< 1%	0.01
v2	Basic	NDR	Stratix V, PCie	3.999	164.20	38%	68%	100%	8%	2.48
			Arria 10, HARP	2.545	162.865	50%	47%	81%	1%	3.90
v3	Basic	SWI	Stratix V, PCie	2.803	191.97	19%	8%	18%	< 1%	3.55
			Arria 10, HARP	3.069	178.12	25%	10%	19%	< 1%	3.24
v5	Advanced	SWI	Stratix V, PCie	0.260	218.15	53%	7%	28%	2%	38.22
			Arria 10, PCie	0.176	201.06	28%	8%	25%	< 1%	56.46
			Arria 10, HARP	0.290	186.81	40%	19%	30%	< 1%	34.27
Dummy	N/A	N/A	Arria 10, HARP	N/A	350.26	23%	6%	14%	0%	N/A

Table 1: Results of executing the Needleman Wunsch kernel versions on the HARPv2 System and how they compare to results in [32]. The values in the *Speedup* column are relative to the kernel Version 0 Stratix V result. The first two rows for kernel Version 5 are both results from [32]. The last row of the table is the result for building a "Dummy" kernel that is simply a kernel that contains no computation.

FPGA resource is utilized relative to the amount available. The execution times presented are the lowest times over 100 runs of the respective kernels. The *Speedup* column is the calculated speedup relative to the Stratix V result from [32] for kernel Version 0.

Comparing results from [32] to those observed from the HARPv2 system, the trends regarding *NDR* and *SWI* kernels reported in [32] also appear here. The applied optimizations to Versions 0 and 1 result in decreases in execution time relative to each kernel's respective runtime. Version 2 executes 5.25 faster than Version 0, and Version 3 executes 270.49 times faster than Version 1. However, our HARPv2 system results, except for Version 2, execute slower than those from [32], despite the Arria 10 FPGA having more resources to use than the Stratix V FPGA.

The biggest contributing factor to this is the amount of resources needed to implement designs. This causes the place and route process of the FPGA to be more difficult, involving more complex routing solutions that drive the maximum possible clock speed down. In almost all cases, the Arria 10 FPGA HARPv2 system uses a larger percentage of its available resources than the Stratix V FPGA does, which has less resources to begin with. In all cases, f_{max} for the Arria 10 is lower than the those for the Stratix V.

This is because of all of the resources necessary to implement the BSP components that interface to the host CPU to the FPGA. The last row in Table 1 shows the resource utilization for a "Dummy" kernel, which is an OpenCL kernel that contains no computation in its function body. We use this as a proxy for the resources required to implement the interface BSP components. As a comparison, consider the *Arria 10, PCIe* result for kernel Version 5, which uses the same FPGA. The percentage of total logic blocks used is 28%, compared to the 23% of logic blocks used just to implement the BSP for the HARPv2 system. Though addressing this shortcoming is compounded by the opacity of the toolflow for the Intel FPGA OpenCL SDK, work done by Sanaullah and Herbordt describe a methodology to isolate the HDL generated from the toolflow [23]. This is done, in part, to classify the common interfaces generated by the tools and either remove unnecessary parts or modify unoptimized parts of the OpenCL-generated HDL to reduce the amount of FPGA resources necessary to build the design and increase performance.

Another inefficiency that is specific to the implementation of kernel Version 5, but applies to both the PCIe and HARPv2 systems, is the way the staircase shift registers are implemented. In the kernel source, this is done by allocating local space for a 2D array and then inferring a shift register from it. Though they are synthesized as a 2D shift register, as shown in Figure 2, only half of it is used. A more efficient approach would be to allocate `PAR` shift registers that are the exact size needed to achieve the buffering effect explained in Section 4.1. However, this is more complex than just allocating a 2D array and inferring a shift register because it involves further tweaks to the OpenCL kernel such as manual unrolling of loops to account for boundary conditions in the algorithm. This problem exemplifies the tradeoff of productivity versus performance.

5.2 Hardware Design Space Search

Table 2 shows the results for sweeping `BSIZE` for kernel Versions 0 and 2, as well as the results for sweeping `BSIZE` and `PAR` for kernel Version 5. As in the previous section, the execution times presented are the lowest times over 100 runs of the respective kernels.

In our experimentation, we define the kernel version design search space for kernel Version 0 as

$$BSIZE = \{64, 128, 256\}.$$

For kernel Version 2, it is

$$BSIZE = \{8, 16\}.$$

For kernel Version 5, the search space is the Cartesian product between

$$BSIZE = \{256, 512, 1024, 2048, 4096, 8192\}$$

and

$$PAR = \{8, 16, 32, 64\}.$$

Kernel version	PAR	BSIZE	Time (sec)	f_{max} (MHz)	Logic	M20K Bits	M20K Blocks	DSP	Build Time (hr:min:sec)
v0	N/A	64	20.530	232.665	30%	16%	28%	1%	11:43:22
		128	13.367	211.77	31%	25%	39%	1%	9:8:36
		256	15.836	153.985	31%	59%	81%	1%	12:56:10
v2	N/A	8	2.545	162.865	50%	47%	81%	< 1%	12:24:00
		16	16.735	182.415	35%	40%	58%	< 1%	7:26:49
v5	8	256	1.011	215.4	27%	12%	21%	< 1%	5:29:05
	8	512	1.035	216.26	27%	12%	21%	< 1%	5:34:20
	8	1024	1.156	213.67	27%	12%	21%	< 1%	14:35:26
	8	2048	1.210	215.26	28%	12%	21%	< 1%	14:22:57
	8	4096	1.227	214.17	27%	12%	22%	< 1%	13:35:46
	8	8192	1.270	213.44	27%	13%	22%	< 1%	6:23:05
	16	256	0.417	200.8	30%	13%	23%	< 1%	9:48:13
	16	512	0.410	209.16	30%	13%	23%	< 1%	9:27:11
	16	1024	0.414	197.86	30%	13%	23%	< 1%	15:25:55
	16	2048	0.437	205.42	30%	13%	24%	< 1%	9:53:23
	16	4096	0.449	196.54	30%	14%	24%	< 1%	6:00:06
	16	8192	1.267	199.84	30%	14%	24%	< 1%	15:21:19
	32	256	0.338	171.02	40%	19%	30%	< 1%	22:30:28
	32	512	0.312	180.27	40%	19%	30%	< 1%	22:18:48
	32	1024	0.298	179.01	40%	19%	30%	< 1%	21:41:14
	32	2048	0.292	186.81	40%	19%	30%	< 1%	18:38:15
	32	4096	0.296	179.5	40%	19%	30%	< 1%	8:04:14
	32	8192	0.297	187.37	40%	19%	30%	< 1%	8:27:33
	64	2048	0.363	117.85	66%	30%	47%	< 1%	31:10:3
	64	4096	0.330	129.04	67%	30%	47%	< 1%	44:54:53
	64	8192	0.332	128.22	67%	31%	48%	< 1%	38:16:34

Table 2: Results for sweeping the `BSIZE` parameter for kernel versions 0 and 2 and the `BSIZE` and `PAR` parameters for kernel version 5.

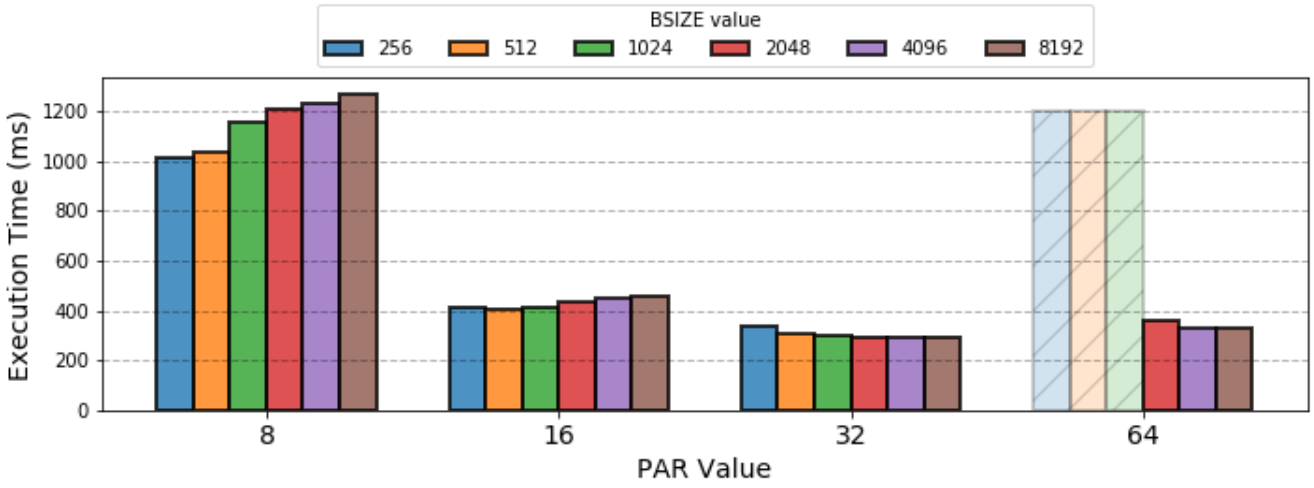

Figure 3: Graphical depiction of execution times for sweeping across hardware parameters `BSIZE` and `PAR` in kernel Version 5. The bars with greyed-out and diagonal lines represent parameter configurations for designs that were unable to be fitted for the FPGA.

In the case of kernel Versions 0 and 2, the upper bound of the search space was determined by the largest value that BSIZE could assume while still being synthesizable by the Intel FPGA OpenCL SDK offline compiler. The upper bounds for kernel Version 5 were determined by the amount of time required to synthesize a design. It must be noted, though, that for `BSIZE = 256, 512, 1024` and `PAR = 64`, the compiler was not able to synthesize a design. Investigating the logs revealed that despite multiple attempts at fitting the design, the routing was too congested and the fitting phase ultimately failed. Slightly larger designs fit on the FPGA, i.e. kernels with `BSIZE = 2048, 4096, 8192`, so we attribute these failures to shortcomings with the offline compiler.

The optimal settings for BSIZE in kernel Versions 0 and 2 reported in [32] were 128 and 64, respectively, for the Stratix V FPGA. The optimal HARPv2 BSIZE for Versions 0 and 2 were found to be 128 and 8, respectively. Thus, the setting matches the optimal setting in [32] for Version 0 but not for 2. The design space for Version 2 on HARPv2 did not include the optimal setting from [32], yet it outperformed [32] by a factor of 1.57 and with a smaller BSIZE. For kernel Version 5, BSIZE and PAR are set to 4096 and 64, respectively, for both the Stratix V and the Arria 10 FPGA to achieve optimal performance in [32]. However, the configuration that was optimal in [32] was not the most performant configuration for the HARPv2 system; the result in [32] is 1.64 times faster. While we expect the best configuration not to align for different FPGAs, this speaks to the portability of kernels designed for FPGAs connected through a PCIe slot versus the HARPv2 system even when the FPGAs are the same. Some of this performance difference can be attributed to the large amount of resources used to implement the CPU/FPGA interface as previously discussed. While the performance difference is relatively small, this result also suggests that further consideration must be given when authoring kernels specific to the HARPv2. This is similar to the claim that OpenCL kernels intended for one type of accelerator will not be the most performant for another type made in [33] when describing GPUs and FPGAs.

The build times of the different configurations of the kernels are also shown in Table 2. Perhaps the most startling result is the amount of time spent buiding kernels for Version 5. The longest build time was for `BSIZE = 4096` and `PAR = 64`, which took nearly two days. In total, it took 14 days to build all the kernels in order to search the design space and find the most optimal kernel. The amount of time it takes to search the design space by brute force necessitates the need for performance models, using facets of the kernel and its estimated resources as inputs, that can be more intelligently searched. To this end, work done by Wang et al. has demonstrated progress in this area by modeling OpenCL workloads on FPGAs for the NDR model [27]. Additionally, it would be beneficial to isolate the parameters of such analytic models which affect performance the most in order to prune the search space of lower weight parameters. Consider Figure 3, which graphically shows the execution times for all the configurations of kernel Version 5.

When PAR is small (i.e., `PAR = 8`), execution time increases as BSIZE increases because the effects of the inefficient staircase register allocation outweighs the benefits of processing a small number of the substitution matrix in a pipeline-parallel fashion. However, as PAR grows, the effects of processing more and more columns in parallel has a greater impact on performance than sweeping the BSIZE parameter. Holding BSIZE to some constant and sweeping the PAR parameter, which has only 4 discrete values, would lead to finding the optimal setting of 32 for PAR. In this case, the range of execution times for the different values of BSIZE is 46 *ms* at 4 days of kernel build time, while the global range is 978 *ms* at 14 days. This then becomes a tradeoff between an approximate answer found quickly versus a precise answer found slowly.

5.3 SVM Performance

Since the kernel built for the runs with explicit reads and writes is the same one used for the runs using SVM, the FPGA resource utilization remains the same between the two. Figure 4 shows the benefit in modifying the host code to use the SVM abstraction, as described in Section 4.3, for kernel Version 5 at the best performing parameter configuration for the HARPv2 system: `BSIZE = 2048` and `PAR = 32`. The execution time reported is the smallest out of 100 runs.

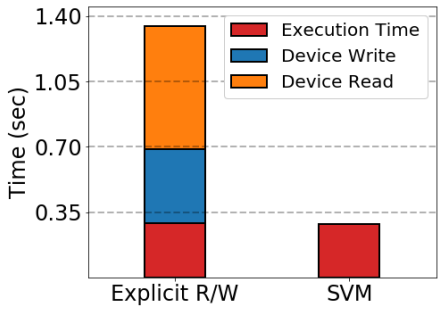

Figure 4: Execution times for kernel Version 5 with `BSIZE = 2048` and `PAR = 32` for host code that enqueues reads and writes explicitly to the device and host code that uses the SVM abstraction.

The left bar shows total amount of time the kernel took to execute, as well as the explicit reads and writes to global memory. Both the explicit reads and writes take longer than the execution of the kernel and increase the running time by a factor of 4. The right bar shows the execution time using the modified host code that uses the SVM abstraction. The time taken to allocate shared buffers was also recorded, but takes 10s of milliseconds and is negligible relative to the execution time. The time taken to explicitly read and write buffers was not recorded in [32], but conservatively assuming that explicit reads and writes execute in one-eighth the time that it does on the HARPv2 system would still have the HARPv2 outperforming the Arria 10 FPGA connected via PCIe slot.

This coherent, low-latency access to shared memory has important implications that require rethinking current paradigms of offloading computation to accelerators. Most commonly for compute-intensive tasks marked for accelerator offload, all data necessary for the computation is moved from from host to device. Since data movement is such an expensive operation, it is beneficial to perform as much computation as possible on the accelerator before shipping the results back to the host. This is the model used in all of the Needleman-Wunsch kernel versions in this work when

using explicit reads and writes. The initial state for the substitution matrix and the entirety of the score matrix are moved to the FPGA. Once all of the substitution matrix has been computed, the updated substitution matrix is moved back to the host for reading.

The tighter integration present in the HARPv2 system, however, would allow for more fine-grained interactions between host and device without the overhead currently of explicitly moving data from CPU to FPGA memory. Huang et al. have investigated the tradeoffs associated in partitioning tasks and data in heterogeneous systems that collaboratively use CPUs and FPGAs attached via a PCIe bus [13]. They find that both partitioning schemes improve the execution time over systems that do not use any kind of collaborative execution. Future OpenCL FPGA kernels targeting the HARPv2 system, then, should take advantage of the low latency communication between shared memory and the FPGA. Additionally, application designers should find ways to collaboratively use the CPU and FPGA for a computation region of interest, instead of relying on one or the other to perform the entirety of that region.

6 CONCLUSION

As Moore's law draws nearer, researchers across disciplines are looking beyond relying on performance increases through faster CPU clock speeds and advances in semiconductor process technologies. FPGAs offer a heterogenous compute solution to this problem by enabling the creation of application-specific hardware that accelerates computation. While the barrier to entry has historically been steep, advances in High Level Synthesis (HLS) are making FPGAs more accessible. Specifically, the Intel FPGA OpenCL SDK allows software designers to abstract away low level details of architecting hardware on an FPGA and allows them to author computational kernels in higher level languages. Furthermore, Intel has developed a system that incorporates both a multicore Xeon CPU and Arria 10 FPGA into the same chip package, as part of the Heterogeneous Accelerator Research Program (HARP), that can be targeted by their SDK.

In this work, we target the second iteration of the HARP platform (HARPv2) using HLS through porting OpenCL kernels written for FPGAs connected via PCIe card. We evaluate their performance against previously reported results, explore the portability of kernels intended for PCIe-connected FPGAs through a hardware design space search, and empirically show the benefits of using the SVM abstraction over explicit reads and writes to the FPGA. Additionally, all artifacts associated with this paper (code and data) are available through WashU OpenScholarship [4].

Through this research, we have also identified many directions for future work. The Hardware Description Language (HDL) level representation generated by the offline compiler may contain many unused hardware components or unnecessary connections to the components of the Board Support Package (BSP). Isolating these and editing/removing them would free up resources and make the FPGA place and route process easier for the offline compiler. Searching the entire hardware design space by brute force is inefficient. The amount of time spent building kernels for evaluation could be shortened by using performance models to represent the kernels. Searching that design space would be faster than building all possible kernel variations. Additionally, developing a method of isolating the parameters of the model that most affect performance would allow for a pruning of the search space, thus shortening the time spent building kernels. Finally, it would be beneficial to target algorithms that can take advantage of the low latency between shared memory and FPGA, as well as research how to author kernels in OpenCL that exploit that low latency.

ACKNOWLEDGMENTS

Supported by NSF grants CNS-1205721, CNS-1527510, CCF-1527692, and CNS-1763503. Thanks to Intel for access to the CPU+FPGA system through the Hardware Accelerator Research Program. Thanks to Alex Hsu of the Texas Advanced Computing Center (TACC) for prompt responses and feedback to our queries regarding the HARPv2 nodes at TACC.

REFERENCES

[1] Fredy Augusto M Alves, Peter Jamieson, Lucas B da Silva, Ricardo S Ferreira, and José Augusto M Nacif. 2017. Designing a collision detection accelerator on a heterogeneous CPU-FPGA platform. In *Proc. of Int'l Conf. on ReConFigurable Computing and FPGAs*.

[2] Amazon Web Services 2019. Amazon EC2 Instance Types. Retrieved January 2019 from https://aws.amazon.com/ec2/instance-types/

[3] Jonathan Bachrach, Huy Vo, Brian Richards, Yunsup Lee, Andrew Waterman, Rimas Avižienis, John Wawrzynek, and Krste Asanović. 2012. Chisel: constructing hardware in a Scala embedded language. In *Proc. of 49th Design Automation Conference*. 1212–1221.

[4] Anthony M. Cabrera and Roger D. Chamberlain. 2019. Exploring Portability and Performance of OpenCL FPGA Kernels on Intel HARPv2: Research Artifacts. https:/doi.org/10.7936/m2yq-a123.

[5] Srihari Cadambi, Abhinandan Majumdar, Michela Becchi, Srimat Chakradhar, and Hans Peter Graf. 2010. A Programmable Parallel Accelerator for Learning and Classification. In *Proc. of 19th Int'l Conf. on Parallel Architectures and Compilation Techniques*. 273–284.

[6] Andrew Canis, Jongsok Choi, Mark Aldham, Victor Zhang, Ahmed Kammoona, Jason H Anderson, Stephen Brown, and Tomasz Czajkowski. 2011. LegUp: high-level synthesis for FPGA-based processor/accelerator systems. In *Proc. of 19th ACM/SIGDA Int'l Symp. on Field Programmable Gate Arrays*. 33–36.

[7] Adrian M. Caulfield, Eric S. Chung, Andrew Putnam, Hari Angepat, Jeremy Fowers, Michael Haselman, Stephen Heil, Matt Humphrey, Puneet Kaur, Joo-Young Kim, Daniel Lo, Todd Massengill, Kalin Ovtcharov, Michael Papamichael, Lisa Woods, Sitaram Lanka, Derek Chiou, and Doug Burger. 2016. A cloud-scale acceleration architecture. In *Proc. of 49th IEEE/ACM Int'l Symp. on Microarchitecture*. 7:1–7:13.

[8] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W Sheaffer, Sang-Ha Lee, and Kevin Skadron. 2009. Rodinia: A benchmark suite for heterogeneous computing. In *Proc. of IEEE Int'l Symp. on Workload Characterization*. 44–54.

[9] Young-kyu Choi, Jason Cong, Zhenman Fang, Yuchen Hao, Glenn Reinman, and Peng Wei. 2016. A Quantitative Analysis on Microarchitectures of Modern CPU-FPGA Platforms. In *Proc. of 53rd Design Automation Conference*. 109:1–109:6.

[10] Thomas Faict. 2018. *Exploring OpenCL on a CPU-FPGA Heterogeneous Architecture*. Master's thesis. Ghent University. https://lib.ugent.be/fulltxt/RUG01/002/495/122/RUG01-002495122_2018_0001_AC.pdf

[11] Clément Farabet, Cyril Poulet, Jefferson Y Han, and Yann LeCun. 2009. CNP: An FPGA-based processor for convolutional networks. In *Proc. of Int'l Conf. on Field Programmable Logic and Applications*. 32–37.

[12] Maya Gokhale, Jan Stone, Jeff Arnold, and Mirek Kalinowski. 2000. Stream-oriented FPGA computing in the Streams-C high level language. In *Proc. of IEEE Symp. on Field-Programmable Custom Computing Machines*. 49–56.

[13] Sitao Huang, Li-Wen Chang, Izzat El Hajj, Simon Garcia De Gonzalo, Juan Gómez-Luna, Sai Rahul Chalamalasetti, Mohamed El-Hadedy, Dejan Milojicic, Onur Mutlu, Deming Chen, et al. 2019. Analysis and Modeling of Collaborative Execution Strategies for Heterogeneous CPU-FPGA Architectures. (April 2019).

[14] Intel-Altera. 2016. Altera SDK for OpenCL Best Practices Guide. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/archives/ug-aocl-best-practices-guide-16.0.pdf

[15] Arpith Jacob, Joseph Lancaster, Jeremy Buhler, Brandon Harris, and Roger D. Chamberlain. 2008. Mercury BLASTP: Accelerating Protein Sequence Alignment. *ACM Trans. Reconfigurable Technol. Syst.* 1, 2 (June 2008), 9:1–9:44.

[16] Zheming Jin and Hal Finkel. 2018. Evaluation of MD5Hash Kernel on OpenCL FPGA Platform. In *Proc. of IEEE International Parallel and Distributed Processing Symposium Workshops*. 1026–1032.

[17] Ryouhei Maeda and Tsutomu Maruyama. 2017. An implementation method of Poisson image editing on FPGA. In *Proc. of 27th Int'l Conf. on Field Programmable Logic and Applications*. 1–6.

[18] Atabak Mahram and Martin C Herbordt. 2012. FMSA: FPGA-accelerated ClustalW-based multiple sequence alignment through pipelined prefiltering. In *Proc. of IEEE 20th Int'l Symp. on Field-Programmable Custom Computing Machines*. 177–183.

[19] Nathaniel McVicar, Chih-Ching Lin, and Scott Hauck. 2017. K-mer counting using Bloom filters with an FPGA-attached HMC. In *Proc. of IEEE 25th Int'l Symp. on Field-Programmable Custom Computing Machines*. 203–210.

[20] Abhinav Podili, Chi Zhang, and Viktor Prasanna. 2017. Fast and efficient implementation of Convolutional Neural Networks on FPGA. In *Proc. of 28th Int'l Conf. on Application-specific Systems, Architectures and Processors*. IEEE, 11–18.

[21] Andrew Putnam, Adrian M. Caulfield, Eric S. Chung, Derek Chiou, Kypros Constantinides, John Demme, Hadi Esmaeilzadeh, Jeremy Fowers, Gopi Prashanth Gopal, Jan Gray, Michael Haselman, Scott Hauck, Stephen Heil, Amir Hormati, Joo-Young Kim, Sitaram Lanka, James Larus, Eric Peterson, Simon Pope, Aaron Smith, Jason Thong, Phillip Yi Xiao, and Doug Burger. 2014. A Reconfigurable Fabric for Accelerating Large-scale Datacenter Services. In *Proc. of 41st Int'l Symp. on Computer Architecuture (ISCA)*. 13–24.

[22] Ahmed Sanaullah and Martin C Herbordt. 2018. FPGA HPC using OpenCL: Case Study in 3D FFT. In *Proc. of 9th Int'l Symp. on Highly-Efficient Accelerators and Reconfigurable Technologies*. 7:1–7:6.

[23] Ahmed Sanaullah and Martin C Herbordt. 2018. Unlocking Performance-Programmability by Penetrating the Intel FPGA OpenCL Toolflow. In *Proc. of IEEE High Performance Extreme Computing Conference*.

[24] David Sidler, Zsolt István, Muhsen Owaida, and Gustavo Alonso. 2017. Accelerating Pattern Matching Queries in Hybrid CPU-FPGA Architectures. In *Proc. of ACM International Conference on Management of Data*. 403–415.

[25] Greg Stitt, Abhay Gupta, Madison N. Emas, David Wilson, and Austin Baylis. 2018. Scalable Window Generation for the Intel Broadwell+Arria 10 and High-Bandwidth FPGA Systems. In *Proc. of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 173–182.

[26] Qing Y. Tang and Mohammed A. S. Khalid. 2016. Acceleration of k-Means Algorithm Using Altera SDK for OpenCL. *ACM Trans. Reconfigurable Technol. Syst.* 10, 1 (Dec. 2016), 6:1–6:19.

[27] Shuo Wang, Yun Liang, and Wei Zhang. 2017. FlexCL: An analytical performance model for OpenCL workloads on flexible FPGAs. In *Proc. of 54th Design Automation Conference*.

[28] Yu Wang. 2018. *Accelerating Graph Processing on a Shared-Memory FPGA System*. Ph.D. Dissertation. Carnegie Mellon University.

[29] Masato Yoshimi, Yasin Oge, and Tsutomu Yoshinaga. 2017. Pipelined Parallel Join and Its FPGA-Based Acceleration. *ACM Trans. Reconfigurable Technol. Syst.* 10, 4 (Dec. 2017), 28:1–28:28.

[30] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. 2015. Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks. In *Proc. of ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays*. 161–170.

[31] Chi Zhang and Viktor Prasanna. 2017. Frequency Domain Acceleration of Convolutional Neural Networks on CPU-FPGA Shared Memory System. In *Proc. of ACM/SIGDA Int'l Symp. on Field-Programmable Gate Arrays*. 35–44.

[32] Hamid Reza Zohouri. 2018. *High Performance Computing with FPGAs and OpenCL*. Ph.D. Dissertation. Tokyo Institute of Technology. arXiv:1810.09773

[33] Hamid Reza Zohouri, Naoya Maruyama, Aaron Smith, Motohiko Matsuda, and Satoshi Matsuoka. 2016. Evaluating and optimizing OpenCL kernels for high performance computing with FPGAs. In *Proc. of Int'l Conf. on High Performance Computing, Networking, Storage and Analysis (SC16)*. 409–420.

[34] Hamid Reza Zohouri, Artur Podobas, and Satoshi Matsuoka. 2018. Combined Spatial and Temporal Blocking for High-Performance Stencil Computation on FPGAs Using OpenCL. In *Proc. of ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*. 153–162.