Algorithm Acceleration on FPGAs using OpenCL

Suneth Samarasinghe
dept. of Computer Engineering
University of Peradeniya
Sri Lanka
imsuneth@gmail.com

Pubudu Premathilaka
dept. of Computer Engineering
University of Peradeniya
Sri Lanka
pubudu.premathilaka@eng.pdn.ac.lk

Wishma Herath
dept. of Computer Engineering
University of Peradeniya
Sri Lanka
wisheslakshan@gmail.com

Hasindu Gamaarachchi
dept. of Computer Science
University of New South Wales
Australia
hasindu@unsw.edu.au

Roshan Ragel
dept. of Computer Engineering
University of Peradeniya
Sri Lanka
roshanr@eng.pdn.ac.lk

***Abstract*—During the past couple of years, GPUs have been frequently utilized in supercomputers to accelerate various types of data processing. However, the high-power usage of these devices remains a bottleneck in deploying large supercomputers. For this reason, Field-Programmable Gate Arrays (FPGA) are a promising alternative to GPUs specifically because of their relatively low power consumption. Typically, FPGA programming requires skills in complex hardware description (HDL). Currently, one of the growing trends in the FPGA domain is high-level synthesis (HLS) frameworks like OpenCL. HLS allows writing programs in high-level languages such as C. Then, the programs are converted by underlying layers to run on heterogeneous systems consisting of CPU, GPU, and FPGA. Several studies have shown that OpenCL with Altera extensions greatly decreases FPGA implementation time and costs in high-performance computing environments. In this paper, we investigate how the OpenCL implementations performed on FPGAs to accelerate complex algorithms that are expensive in terms of execution time, hardware resources, energy consumption, how they are capable of alleviating computational problems and achieving significant other performance improvements under the areas of bioinformatics, molecular dynamics, machine learning, deep learning, and other computationally intensive applications.**

I. Introduction

This modern era would be most affected by High-Performance Computing (HPC) [1], which focuses on the latest technologies to satisfy the never-ending demand for higher performance and power consumption. This has led the HPC world to procure specialized accelerators in the last few years. The most common accelerator to be used for HPC applications has so far been GPUs.

The Energy consumption of GPUs is usually high and it can be up to 300 watts [1], and advances in GPU power quality are approaching their limits. The key restrictions in the design and usage of large HPC machines are power consumption and reliability, and certain power and cooling limitations are placed on the usability of GPUs in large supercomputers [2]. One of the accelerators that have appeared recently as more power-efficient alternatives to GPUs is FPGAs.

Traditionally, FPGA programs are developed using Hardware Description Languages (HDL), built on a very different programming paradigm compared to traditional software programming languages such as C and Fortran. This problem has always been a huge barrier to the acceptance of FPGAs by software programmers until High-Level Synthesis (HLS) [1] tools are built to make FPGAs available for them to define their FPGA architecture in software programming languages.

Since the advent of HLS, many such tools have been developed and adapted to different areas where performance improvement and energy efficiency are expected. Sudden improvements in the HLS environment also have arisen, with official HLS tools being directly produced and financed by FPGA suppliers, leading FPGAs to be more commonly embraced by software programmers. OpenCL is one such open-source and royalty-free framework [1].

The remaining sections discuss the background of HPC with the advent of high-level programming of FPGAs using OpenCL. Also, we present how different algorithms related to several areas of applications have been accelerated using FPGAs and OpenCL. Finally, we demonstrate a comparison for each chosen algorithms with their related work.

II. Background

A. High-Performance Computing (HPC)

Most of the modern world applications are based on computation-intensive algorithms. These algorithms are deployed in different fields such as biotechnology, image processing, machine learning, molecular dynamics, multimedia, wireless communication, digital signal processing and many more. The primary objective of optimizing computation-intensive algorithms is to reduce the time it takes to complete the computation. This problem aggravates for multidimensional optimization where the search space increases exponentially. And it is also called as "the curse of dimensionality" [3]. In the following paragraphs, we review some of the challenges of deploying these algorithms.

In biotechnology high throughput of biological data sets are generated. These data sets are complex and have large dimensions which lead to the requirement of higher computational power. Microarray is the fundamental instrument in modern cancer research [4] but, the analysis of microarrays is computationally expensive. For example, the K-means clustering algorithm, which is used in microarray analysis is an iterative algorithm requiring long computational time in general-purpose processors.

Image processing and computer vision are natural applications for HPC. Therefore, it is a well-researched area of fine-tuning vision-related algorithms to run on accelerators like FPGAs. The performance bottleneck comes when the image size gets higher with the resolution. The example present in [5] illustrates that at least 66 million operations per second are required to process a standard 720p video stream at 24 frames per second which not even full HD.

In the field of Digital signal processing (DSP), other than computational speed, it is crucial to satisfy higher demand of inputs and outputs imposed by the DSP system [6]. Most of the DSP systems limited by the multiplication operation. Therefore, the rate of multiplication by the system has to maximize. It can achieve by implementing multiplier as a fully parallel array multiplier or a fully bit-serial multiplier.

The most common approach to achieve better performance is by assigning the computationally intensive task to hardware and exploiting the parallelism in the algorithm [7]. Field Programmable Gate Arrays (FPGAs) have proved an effective platform for the implementation of these algorithms. FPGAs are in-between general-purpose processors and ASICs (application-specific ICs) on the spectrum of processing elements [8].

B. High-Level Synthesis

Nowadays, for many high-performance applications like medical imaging [9], molecular dynamics [10], Field Programmable Gate Array (FPGA) devices have become a solution of choice.

Historically, hardware developers used hardware description languages (HDL), programming environments like Verilog and VHDL to program hardware at register transfer level (RTL) [11]. When the application gets large, this approach can be complex and frustrating even with a proper structure of an implementation. The time to design, verify, and optimize (time-to-market) an application using RTL is significant and requires previous experience in hardware design, which implies increased development cost. In the field of bio-informatics, the algorithms evolve rapidly and thus rewriting those changes in HDL is not easy.

This forced developers to come up with high-level synthesis (HLS) tools like Intel OpenCL (Open Computing Language) HLS and Xilinx Vivado HLS [12]. These tools provide the ability to write applications in high-level programming languages such as C/C++ and SystemC and then generate the RTL design of the program to support hardware like FPGAs. HLS reduces the time-to-market and increases the productivity of the developers by taking the overhead of deciding microarchitectural detail of the FPGA design.

C. Intel FPGA SDK for OpenCL

In [13], it has been shown that the OpenCL computing paradigm is a viable design approach for high-performance applications on FPGAs, and it is a framework for parallel programming and includes a language, API, libraries, and a runtime system to support software development. OpenCL is developed to allow parallel computation to accelerate, addressing a wide range of platforms [14]. The programs written in OpenCL can then be converted to RTL designs to support a wide variety of platforms.

Moreover, debugging and verification of hardware designs becomes easier relative to the developments in the RTL level because of the C-level simulation of FPGA designs. However, one downside of using HLS for FPGA programming is the lack of analysis tools available, unlike CPU or GPU programming where there are tools such as VTune and NSight for analyzing purposes. Therefore, it is hard to identify the cause of the bottleneck or performance problem at the hardware level of FPGAs.

OpenCL platform model[1], an abstract hardware model for devices (Figure 1). One platform has a Host and one or more Devices connected to the host. Each Device may have multiple compute units with multiple processing elements(PEs).

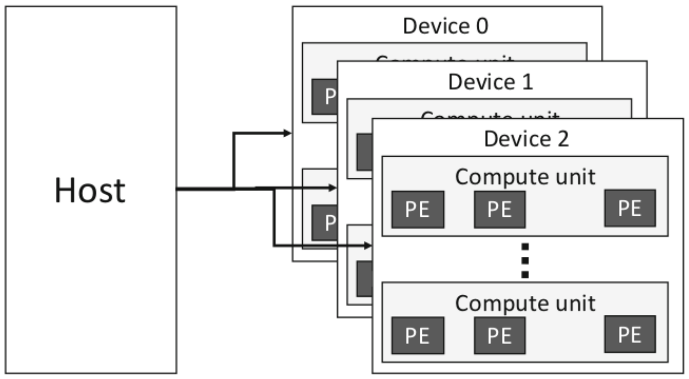


Fig. 1. OpenCL Platform Model

An application with OpenCL consists of two parts: the host and kernel. The host program is used to program the CPU while the kernel is responsible for the computation of FPGA.

The host program performs the following tasks,

- Obtain an OpenCL platform and devices.
- Create a context and a command queue.
- Program the FPGA.
- Allocate memory.
- Transfer the input data from the host to the device.
- Execute the kernel.
- Transfer the output results from the device to the host.
- Release the allocated memory.

The execution model (Figure 2) shows the communication mechanism between the host and devices in the context environment. The host submits work to devices and manages the workload in the context using the OpenCL API platform layer. The command queue is the communication media which the host query the commands such as read, write, and execute.

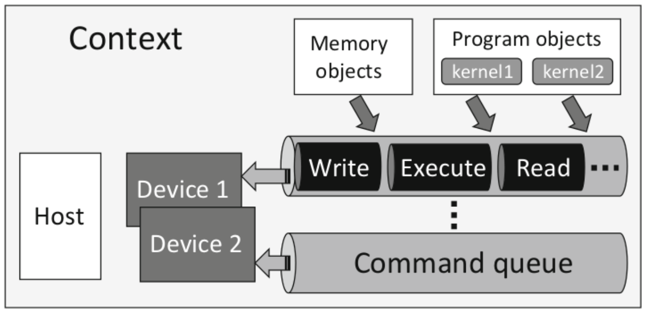


Fig. 2. OpenCL Execution Model

OpenCL for FPGA uses two types of kernels, namely ‘Single work item’ kernels and ‘NDRange kernels’[1] (Figure 3). In a single work item kernel, there is only one work item while NDRange kernel has multiple work items. The Single work item kernel shares data among multiple loop-iterations by using a private memory while NDRange kernels share data among multiple work-items by using local memory.

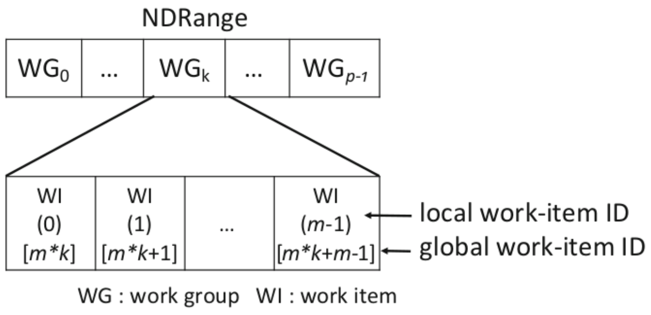


Fig. 3. OpenCL Kernel Programming Model

In[1], it is emphasized loop unrolling, optimizing floating-point operations, optimizing fixed-point operations, optimizing vector operations as common optimization techniques for both single work-item kernels and NDRange kernels.

The memory hierarchy of OpenCL is shown in Figure 4. The host memory is accessible only to the host. The global memory is accessible to both the host and the device. Constant memory is read-only and only accessible to the device. Each work group has a local memory shared by each work item and a work item has its own private memory.

Figure 5 illustrates the schematic diagram of the Intel FPGA SDK for the OpenCL programming model.

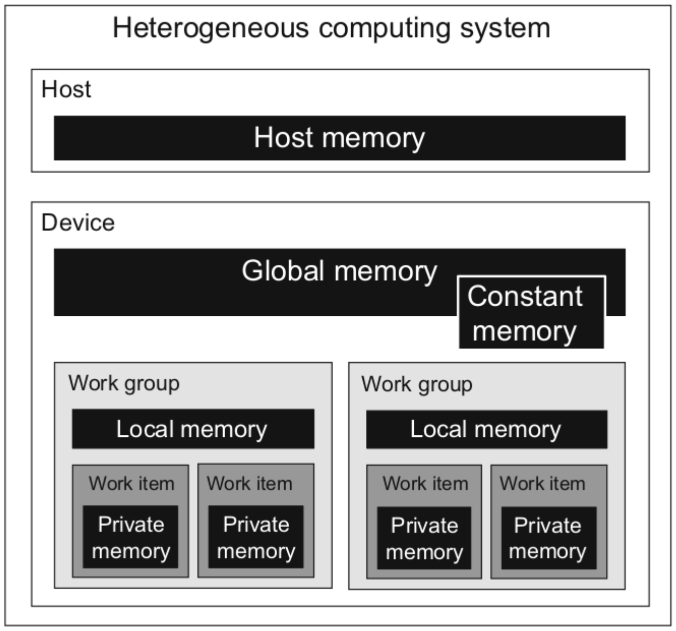


Fig. 4. OpenCL Memory Model

III. Related Work

A. Smith-Waterman Algorithm on FPGA using OpenCL

Smith-Waterman (SW) [16] algorithm is a widely used pairwise sequence alignment algorithm that finds the best possible aligned subsegment in a pair of sequences. Accelerating SW is a great challenge in the field of high-performance computation. Therefore, FPGA implementation of SW is a well-researched area. In [17], Rucci et al. have presented SW implementation which is capable of aligning DNA sequences of unrestricted size for Altera Stratix V using OpenCL. In this work, the kernel is implemented using the task parallel programming model. The alignment matrix is divided into verticle blocks (**BW** means Block Width). In a row-by-row manner, each block is computed from top to bottom and left to right. This approach supported by OpenCL has improved the data locality and has reduced the memory requirement for block execution. A large number of operations per clock cycle is performed using loop instruction pipelining. To avoid read-write dependencies in global memory separate buffers are used. In their experimental result, as shown below, it can be observed that using smaller data types for kernel implementation has increased the performance as well as it has reduced the resource consumption.

In Table I, Adaptive logic modules (ALMs), Registers (Regs), RAM blocks, Digital Signal Processor Blocks (DSPs) are used as resource measurements including different lengths of DNA sequences.

In the above table, if we consider **int_bw256** and **short_bw256**, it is clear that **int** kernel has consumed nearly 0-0.28 resources than **short**, as well as performance increase,

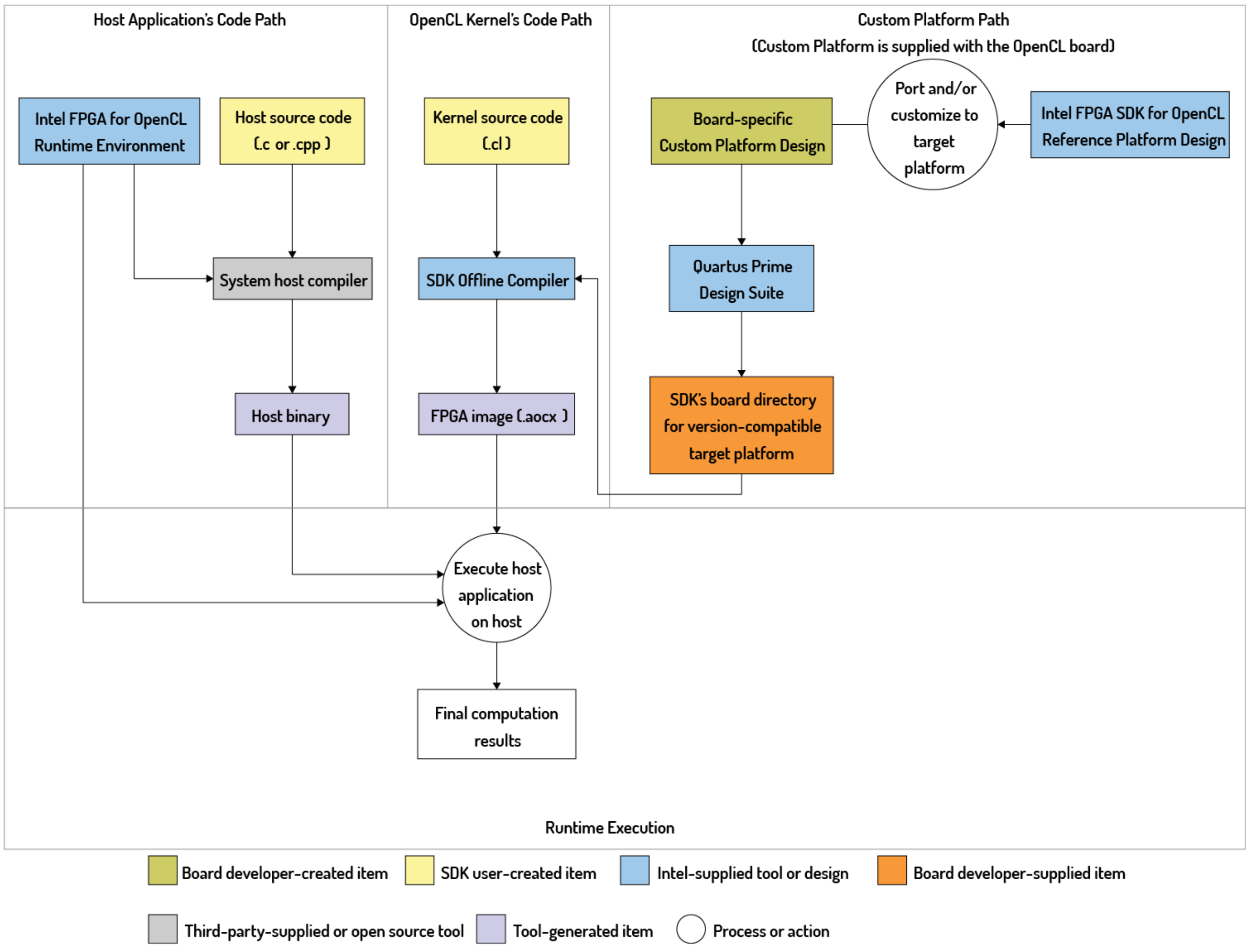

Host Application's Code Path
OpenCL Kernel's Code Path
Custom Platform Path
(Custom Platform is supplied with the OpenCL board)
Intel FPGA for OpenCL Runtime Environment
Host source code (.c or .cpp)
Kernel source code (.cl)
Board-specific Custom Platform Design
Port and/or customize to target platform
Intel FPGA SDK for OpenCL Reference Platform Design
System host compiler
SDK Offline Compiler
Quartus Prime Design Suite
Host binary
FPGA image (.aocx)
SDK's board directory for version-compatible target platform
Execute host application on host
Final computation results
Runtime Execution
Board developer-created item
SDK user-created item
Intel-supplied tool or design
Board developer-supplied item
Third-party-supplied or open source tool
Tool-generated item
Process or action

Fig. 5. Schematic diagram of the Intel FPGA SDK for OpenCL programming model

Kernel	int_bw256	short_bw256
Integer Type	int (32 bits)	short (16 bits)
BW	256	256
ALMs	69%	50%
Regs	12%	10%
RAM	25%	21%
DSPs	0%	0%

TABLE I: Resource usage comparison for different OpenCL kernel implementations

is around 1.22. The latest version of this project [18] which has finetuned previous work and here Intel Arria 10 has been used. There is a significant performance improvement (3-4 X).

Another implementation of SW algorithm with OpenCL and FPGA [19] is presented by Rucci et al. Their work covers protein sequence alignment and testing with real amino acid datasets. In this work, SW kernel has exploited inter-task parallelism. Here, they have utilized SIMD (Single Instruction Multiple Data) vector capability available in the FPGA. Therefore, instead of using one sequence at a time, multiple sequences are aligned at a time. It is emphasized that the allocation of 64-byte host side buffers has improved the data transfer efficiency because Direct Memory Access (DMA) takes place to and from the FPGA.

GCUPS (billion cell updates per second) is a commonly used performance measurement in SW [19]. Performance is evaluated considering different implementations in the degree of data parallelism

- Scalar: Basic code without optimizations
- SIMD version: Implemented using data-level parallelism. **int4**, **int8**, and **int16** vectors are used.

From Table II the vectorized approach has performed better than the scalar version implementation. SMID version used **int16** is nearly 11 times faster than the scalar approach.

In [20], Sirasao et al. have presented FPGA and OpenCL-based acceleration to the SW algorithm. They have benchmarked performance per watt on different hardware platforms including CPUs, GPUs, and FPGAs. Also, it presents perfor-

Version	Performance (GCUPS)
Scalar	3.41
int4	18
int8	23.66
int16	38.9

TABLE II: Performance comparison for different OpenCL kernel implementations

mance tradeoff using OpenCL based programming environment.

Using Systolic array is a well-known approach for computing the scoring matrix in SW algorithm [21]. In [20], this approach has been utilized to achieve better performance. Table III demonstrates the performance of the system with the number of systolic cells.

#Systolic Cells	GCUPS/Kernel	Max-Kernels	Total GCUPS
8	0.8	153	120.0
16	1.6	80	128.1
32	3.2	42	135.4
64	5.8	21	122.3
128	8.7	11	98.9

TABLE III: Theoretical GCUPS vs Kernel sizes

With the increasing number of systolic cells, GCUPS per kernel value is also increased. In this implementation, the peak performance is recorded when using 32 systolic arrays. Therefore, it is highlighted that increasing single kernel performance using many systolic cells does not increase the overall system performance. To increase the overall system performance, it is important to optimize data structures, memory hierarchy, and kernels as well.

B. High-Performance Stencil Computation on FPGA using OpenCL

Stencil computations are one of the most important types of algorithms in High-Performance Computing (HPC) that are widely used applications in the fields of weather, wave, seismic, and fluid simulations, image processing, and convolutional neural networks. It is a grid-based iterative computation method with a large number of parallel operations (ex: Jacobi-2D algorithm). The stencil computation pattern involves updating elements in a multidimensional array according to a relationship based on neighboring cells called a stencil. Here is an example of first-order 2D and 3D star-shaped stencils.

Optimization and tuning them remains challenging for most programmers. The arrangement of stencils exhibits the ability to compute cells in the same iteration in parallel which is called “cell-parallel” computation. But, cell-parallel computation is not suitable for FPGAs due to their smaller external memory bandwidth. Stencil computations are generally memory-bound since the grid size is usually very large and accessed from external memory in each iteration. But this computation pattern shows a good spatial and temporal locality which allows for optimizations with significant reductions in memory usage.

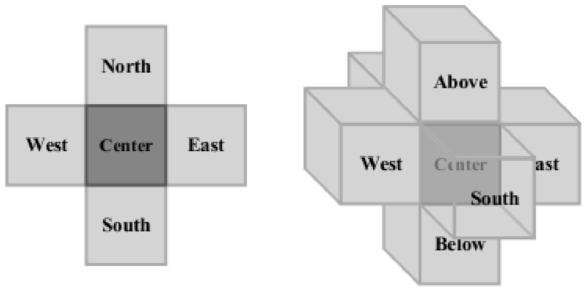


Fig. 6. First-order 2D and 3D stencils

In 2017, Waidyasooriya et al. propose an FPGA platform using OpenCL for stencil computations [22] using iteration-parallel computation where multiple iterations are processed in parallel. With that, they propose an optimization methodology to find the optimal architecture for a given application [22]. They have achieved higher processing speed relative to multicore CPU and GPU implementations and more than 60% of the peak performance given by FPGA.

Moreover, pipelining is used to achieve parallelism since the usual synchronization mechanism using barriers costs a lot of hardware and decrease the performance. They manipulate AOC (Altera Offline Compiler) to analyze loops and automatically generate pipeline stages. Separate hardware is created by AOC for each conditional branch and such hardware for each operation helps to run them parallelly.

They have evaluated on two FPGA boards, three GPUs, and two CPUs with the stencil computation architecture using the fastest implementation out of both temporal blocking and non-temporal blocking methods. These results are shown in Table IV.

In [23], Wang et al. used a similar approach to bridge and enable data sharing between neighboring tiles through using pipes solving the redundant computation and unbalanced workload problems. They proposed a new heterogeneous architecture design for stencil computations to improve performance with saved FPGA resources. Further, they developed a performance model to determine optimal stencil accelerator design parameters and proposed a framework to automatically optimize, synthesize stencil computations onto FPGAs. They achieved a 1.65X performance increment compared to state-of-the-art with fewer hardware resources.

For the validation (Table V) of the optimization framework, they used the Alpha Data ADM-PCIE-7V3 board with a Xilinx Virtex-7 FPGA and 16GB device memory connected to a host via a PCI-e 3.0 X8 interface. They used Xilinx SDAccel 2016.2 as the OpenCL toolchain to synthesize OpenCL kernels onto FPGA. OpenCL based stencil benchmarks from Polybench, Rodnia, and Parboil benchmark suites were used. Overall, the average performance speedup for the proposed Heterogeneous design 1.65X. Also, for Jacobi-2D, their designs eliminate 17% of redundant computation and 6% of memory transfer time of the overall execution time in the

		FPGA	GPU	Multicore CPU
		DE5	GTX960	i7-4960x
Specifications	Number of Cores	-	1024	6
	Core Clock Freq(MHz)	≈270	1127	3600
	Memory Bandwidth(GB/s)	25.6	112	51.2
	Peak Performance(Gflop/s))	196	2308.1	345.6
Processing Time(s)	Laplace Equation	45.3	111.7	260.2
	2-D 5-Point Jacobi	139.2	113.0	281.5
	2-D 9-Point Jacobi	259.8	153.1	419.9
	2-D FDTD	20.5	41.3	290.7

TABLE IV: Performance comparison between FPGA, GPU and CPU

baseline design. 9% of the waiting time of the synchronization barrier was eliminated because of the workload balancing technique using pipes.

Benchmark	Optimization	Pref.
Jacobi-1D	Baseline	1
	Heterogeneous	1.19
Jacobi-2D	Baseline	1
	Heterogeneous	1.58
Jacobi-3D	Baseline	1
	Heterogeneous	2.05
HotSpot-2D	Baseline	1
	Heterogeneous	1.35
HotSpot-3D	Baseline	1
	Heterogeneous	1.97
FDTD-2D	Baseline	1
	Heterogeneous	1.48
FDTD-3D	Baseline	1
	Heterogeneous	1.90

TABLE V: Experimental results of stencil benchmark suite

In [24], Jia et al. have Optimized 1D convolution, 2D convolution, and 2D Jacobi iteration kernels for both Single-Task and NDRange modes. They were able to gain 7.1X and 3.5X speedup factor for the Sobel and Time-domain FIR filters than Altera design examples.

The evaluation (Figure 7) of their implementations was done on a Terasic's DE5-NET board, which includes 2-bank 4GB DDR3 device memory and an Altera Stratix V GX FPGA. The Altera OpenCL SDK v16.0 is used to compile the OpenCL code for FPGAs.

C. K-Nearest Neighbor algorithm (KNN) on FPGA using OpenCL

K-Nearest Neighbor Algorithm(KNN) is one of the most popular machine learning algorithms [25] and it is widely used in pattern recognition. Due to high computational complexity for large datasets, the KNN algorithm has become popular in the field of high-performance computing.

In [26], Pu et al. have proposed a new solution to speed up the KNN algorithm on FPGA based heterogeneous computing system with OpenCL. They have introduced a specific bubble sorting algorithm based on FPGA's parallel pipeline structure to optimize the KNN algorithm. In this implementation, they have used two kernels namely 'Distance calculation' kernel and 'Distance Sorting' kernel.

The distance calculation kernel is parallelized based on data parallelism. Using this approach, they have achieved maximum concurrency of distance calculation for each work item. Moreover, this kernel is contained an internal loop that has been unrolled eight times. Loop unrolling used less memory than a full replication and also it increases the throughput. The Distance sorting kernel has vectorized twice maximum resource utilization on FPGA. Vectorization has been limited to two because the bandwidth of the global memory access is limited.

Table VI illustrates performance for each kernel on CPU, GPU, and FPGA. The CPU is an Intel Core i7-3770K running at 3.5GHz. The GPU is an AMD Radeon HD7950 with 28 compute units and a maximum working frequency 900MHz. The FPGA board is a Terasic DE4 with a Stratix IV 4SGX530.

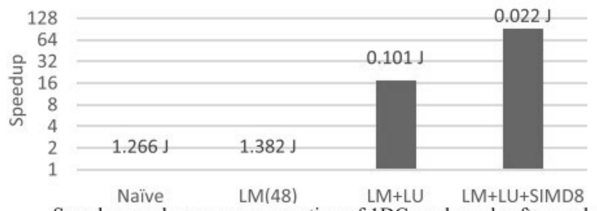
Platform	CPU	GPU	FPGA
Feature Size/nm	22	28	40
Runtime/ms	10211.05	24.85	69.12
Objects/s	1.96	804.96	289.34
Speedup	/	410	148
Power/W	130	200	24
Objects/J	0.015	4.024	12.056
EER	/	268	804

TABLE VI: Performance comparison between CPU, GPU and FPGA

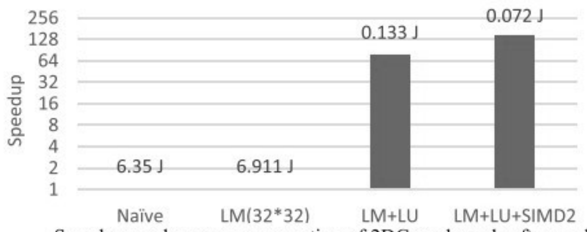
In this test 20 query objects are used. The GPU accelerated KNN algorithm by 410 times the speed of the 4-threads CPU implementation, while FPGA achieved 148 times. Though, the computation speed is fast in GPU, if performance averaged to Joule, FPGA becomes the best.

When comparing the power consumption CPU implementation could merely classify 0.015 query objects per Joule and GPU achieved 4.024, while FPGA 12.056. The energy-efficient ratio (EER) in FPGA is 3 times better than the GPU.

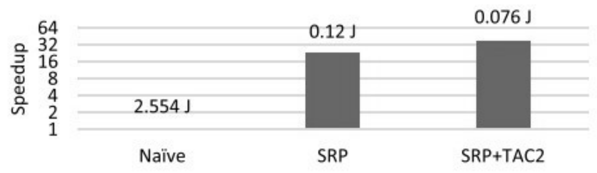
Two different implementations of the energy-efficient approach for the KNN algorithm is presented [27] by Muslim et al. Furthermore, they have compared the performance between GPU and FPGA implementations of the same algorithm.



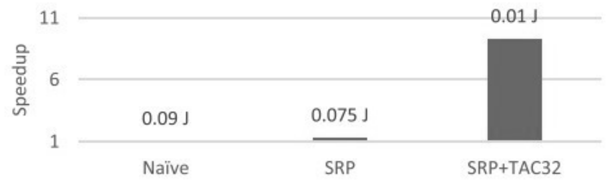
Speedups and energy consumption of 1DConv kernels after applying optimizations in the NDRange mode.



Speedups and energy consumption of 2DConv kernels after applying optimizations in the NDRange mode.



Speedups and energy consumption of 2DConv kernels after applying optimizations in the Single-Task mode.



Speedup and energy consumption of 2D Jacobi Iteration kernels after applying optimizations in the Single-Task mode.

Fig. 7. Speedups and energy consumption of 1D, 2D Conv kernels

In the first approach, both sorting and nearest neighbor identification performed by the host. It uses only global memory and because of independent data usage algorithm extremely parallelizable.

In the second approach, they have implemented two kernels to calculate distances and to find k-smallest distances and return their indices at the end of execution. Moreover, in this implementation, they have utilized a memory optimization technique offered by SDAccel. SDAccel is a Xilinx development environment for synthesizing OpenCL kernels to be executed on Xilinx FPGA devices [28].

The experiment set up includes three hardware platforms. Device 1: NVIDIA GeForce GTX960 GPU with 1024 cores and a maximum operating frequency of 1178MHz. The device has about 2GB GDDR5 of global memory, with 112GB/s of memory bandwidth. Device 2: NVIDIA Quadro K4200 GPU with 1344 CUDA cores and a maximum clock frequency of 784MHz. The device has about 4GB of GDDR5 global memory, with 172.8GB/s of memory bandwidth. Device 3: Alpha data ADM-PCIE-7V3 FPGA board with a Virtex-7 690t. The global memory consists of two DDR3 memories with 21.3GB/s of bandwidth. K value is set to 5 and 300000 number of data points used in the test. Table VII shows the performance of the first approach.

Parameters/Devices	FPGA	GTX960	K4200
Device Time/ms)	1.24	0.04	0.05
Sort Time(Host)/ms	4	3	3
Total Execution Time/ms	5.24	3.04	3.05
Power(Device)/W	0.422	120	108
Energy(Device)/mJ	0.523	4.4	5.6

TABLE VII: Performance analysis of implementation I

Both GPUs perform better than the FPGA because of their higher DRAM access bandwidth. In terms of power and energy consumption, FPGA implementation has outperformed. Since sorting is done by the host, it has impacted the performance of the FPGA implementation. Table VIII shows the performance of the second approach.

Parameters/Devices	FPGA	GTX960	K4200
Total Execution Time/ms	1.23	930	3110
Power(Device)/W	3.136	120	108
Energy(Device)/J	0.0039	111.6	335.88

TABLE VIII: Performance analysis of implementation II

In this approach FPGA implementation is the fastest and still, it consumes lesser power and energy. It has performed 7-times faster than the first approach. Since multiple kernels execute sequentially on the GPUs and it shares only global memory it has underperformed compared to FPGA which executes kernel parallelly. Also using SDAccel, it automatically maps global arrays used solely to inter kernel communication to the on-chip block RAMs.

D. Convolutional Neural Networks (CNN) on FPGA using OpenCL

Convolutional Neural Networks have the most popular deep learning architecture majorly in the field of computer vision. It is adapted in many applications such as image classification, facial detection, video analysis, and speech recognition. Since they consist of multiple computationally intensive convolutions (to extract features from the input data) and fully-connected layers (all the input features are connected to all of the output features through synaptic weights), CNNs are usually accelerated by GUPs with high power consumption. But it is challenging to apply them for real-time applications with the requirement of low power consumption. Recent studies on accelerating CNNs on FPGAs especially with high-level synthesis have shown the advantage of reconfigurability and energy efficiency and fast turn-around-time over GPUs.

In [29], Suda et al. proposed a systematic design space exploration methodology to maximize the throughput of an OpenCL based FPGA accelerator for a given on-chip memory, registers, computational resources, and external memory bandwidth. They implemented a CNN with fixed-point operations on FPGA using OpenCL and identified critical design variables that affect the throughput and execution times were modeled and validated as a function of those variables. They proposed and demonstrated a systematic way to minimize the total execution time of large scale CNNs: AlexNet [30] and VGG [31] on FPGAs.

Conventional CNN models are trained using CPUs and GPUs and it is difficult to implement them on an embedded platform since they consume a significant amount of storage, external memory bandwidth. In their research, they state the reduction of data precision of the weight/data can reduce the storage requirement as well as the energy for memory transfer without any impact on the accuracy. Figrure 8 depicts the accuracy variation of AlexNet and VGG CNNs with precision of convolution weights.

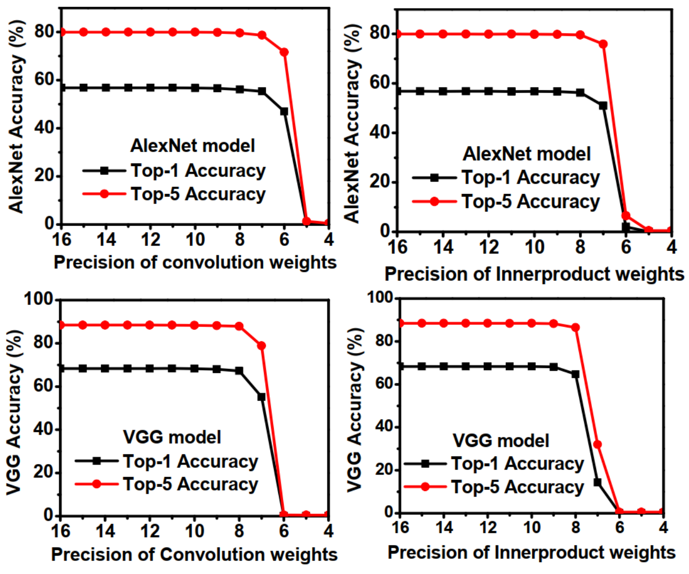


Fig. 8. AlexNet and VGG CNN accuracy variation with precision of convolution weights

The validation (Table IX) of the proposed optimization framework was done by accelerating AlexNet and VGG CNN models on two Altera Stratix-V based boards using OpenCL with fixed-point operations using an 8-bit for convolution.

In [32], Zhang et al. propose an analytical performance model to perform in-depth, quantitative analysis on resource requirements and performance of CNN classifier kernels and available resources on modern FPGAs. Further, they propose a new kernel design to address the key performance bottleneck of chip memory bandwidth that was identified by applying the model to analyze VGG CNN to optimally balance between computation, on-chip and off-chip memory access.

	FPGA	Classification Time/Image(ms)	Throughput(GOPS)
AlexNet	P395-D8	20.01	72.4
	DE5-Net	45.7	31.8
	CPU	191.9	7.6
AlexNet	P395-D8	262.9	117.8
	DE5-Net	651.2	47.5
	CPU	1437.2	21.5

TABLE IX: Classification time/image and overall throughput

They have verified the effectiveness of the proposed model and were able to achieve the highest performance, energy efficiency, and performance density relative to state-of-art OpenCL FPGA CNN implementations.

In their proposed kernel design, they propose a novel 2D interaction between PEs and local memory (Figure 9, 10) using multicast connections between PEs and local memory instead of conventional unicast connections that allows re-usage of the data by sharing. Moreover, the work-items are always scheduled along the lower dimension by the current OpenCL dispatcher or 1D interconnections between PEs and local memory. To support the above-mentioned 2D interaction, they have proposed to dispatch work-items in a two-dimensional manner (Figure 11). Also, they have designed a line buffer between local and external memory to flatten and rearrange data to improve the re-usage of on-chip data. Further, the ping-pong mechanism has been used to fill the line buffer as a remedy for the external memory access latency.

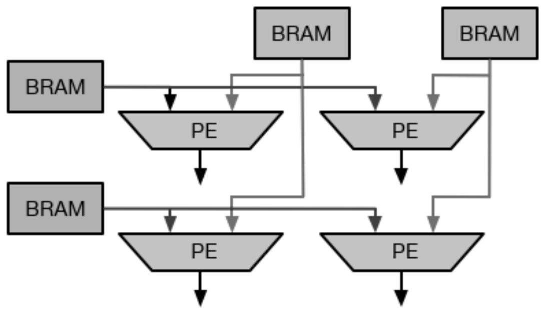


Fig. 9. Proposed two-dimensional PE-to-local memory interconnection

For the evaluation (Table X), they have used an Altera Arria10 FPGA development board which consists of Arria10 GX1150 FPGA and a 1GB DDR4 SDRAM module with 12GB/s bandwidth with OpenCL SDK for FPGA and Quartus Pro 16.

In [33], Wang et al. introduced and demonstrated PipeCNN which is an efficient FPGA accelerator that is open for researchers to be implemented on a variety of FPGA platforms with reconfigurable performance and cost. It includes a set on OpenCL kernels namely Convolutional kernel (Figure 12), Data mover kernel (Figure 13), and other kernels integrated using Altera's OpenCL extension channels.

		Proposed		Baseline		Speedup
Layer	Number of Operations(GOPs)	Duration(ms)	Perf. (GOPs/s)	Duration(ms)	Perf. (GOPs/s)	
CONV1	3.87	3.5	1098.04	21.3	181.6	6.03X
CONV2	5.55	4.4	1232.04	26.8	207.11	5.95x
CONV3	9.25	6.8	1329.57	41.4	223.35	5.94x
CONV4	9.25	7.4	1347.77	45.0	205.25	6.56x
CONV5	2.31	1.8	1223.94	10.9	210.73	5.82x
CONV TOTAL	30.69	23.9	1284.94	145.5	207.7	6.18x
FC6	0.029	4.1	7.25	4.8	6.09	1.19x
FC7	0.034	5.2	6.58	6.2	5.52	1.19x
FC8	0.0082	1.8	4.50	2.1	3.78	1.19x
FC TOTAL	0.073	11.1	6.6	13.2	5.52	1.19x
TOTAL	30.76	35.5	866	158.8	196	4.41x

TABLE X: Experiment results of CNN accelerator based on VGG

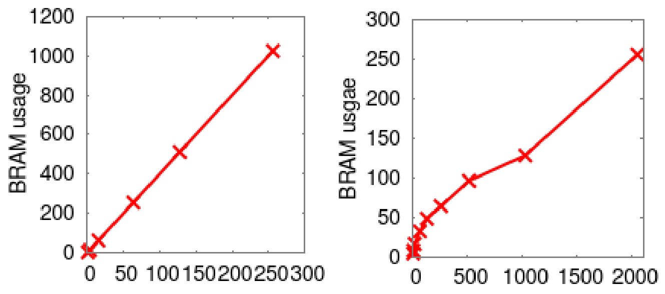


Fig. 10. BRAM usage for (a) traditional one-dimensional and (b) proposed two-dimensional interconnection between PEs to local memory when varying the number of PEs using Arria10 AX1150 FPGA

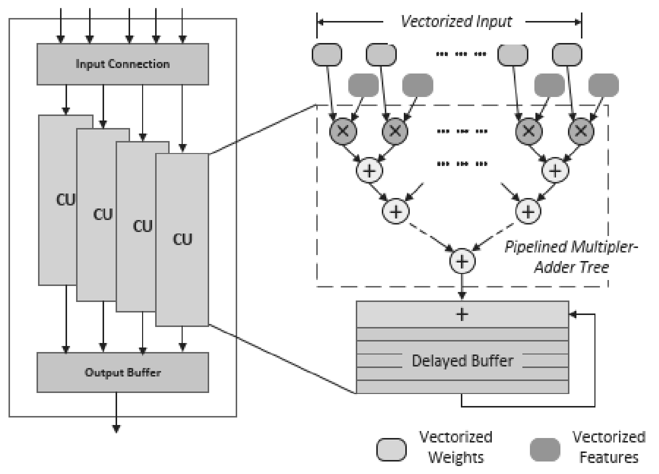


Fig. 12. The hardware architecture of the convolution kernel

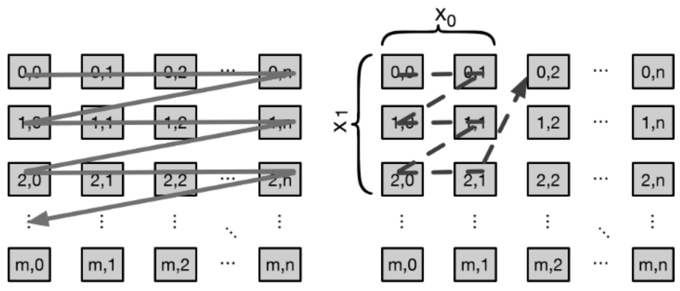


Fig. 11. Conceptual diagram illustrating (a) traditional one-dimensional dispatcher and (b) proposed two-dimensional dispatcher

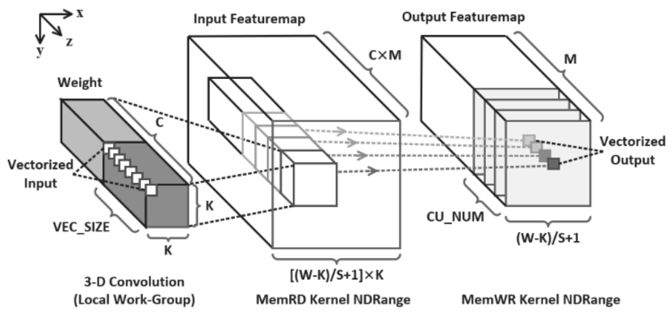


Fig. 13. Data and work-item mapping scheme of the data mover kernels.

Throughput optimization is done by data vectorization and parallels CUs. Optimizations of bandwidth are achieved by introducing a sliding-window data buffering scheme (Figure 14) and fixed-point arithmetics are used instead of floating-point to reduce memory bandwidth requirements and hardware costs.

The demonstration and evaluation (Table XI) were done on three FPGAs consisting of Cyclone-V SEA5 SoC, Stratix-V GXA7, and Arria-10 AX115 using OpenCL SDK 16.1, and processing speed and power consumption were measured for CNN-based image classification (AlexNet with 8 layers and VGG with 16 layers). The host was powered with Intel i5-4690K CPU and 64GB memories.

E. Molecular dynamics applications on FPGA using OpenCL

Molecular dynamic (MD) is the area of computer simulations to analyze the physical behavior of atoms and molecules

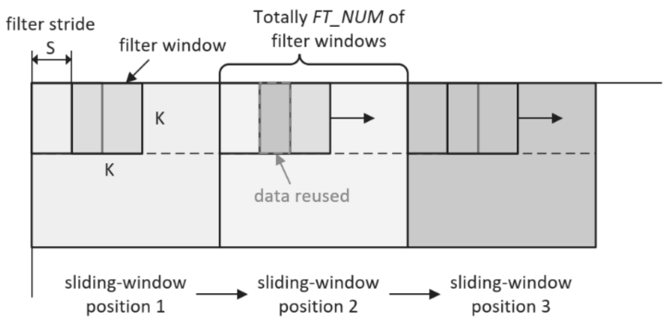

filter stride
S
filter window
Totally *FT_NUM* of filter windows
K
K
data reused
sliding-window position 1
sliding-window position 2
sliding-window position 3

Fig. 14. Sliding-window-based data buffering scheme

Platform	FPGA Type	Resource Capacity	Resource Consumed	Execution Time(ms)		Frequency(Hz)	Board Power(W)
				AlexNet	VGG-16		
DE1-soc	Cyclone-V SEA5	85K LEs 87 DSPs	45K LEs 68 DSPs	140	1928	122	2.1
DE5-net	Stratix-V GXA7	622K LEs 256 DSPs	122K LEs 247 DSPs	15	254	198	27
DE5a-net	Arria-10 GX1150	1150K LEs 1518 DSPs	322K LEs 683 DSPs	5	110	218	26

TABLE XI: Summary of the measured performance, cost and power consumption on different platforms

in space. The simulation is driven by the numerical results given by relatively applying classical Newtonian dynamic equations to atoms or molecules. These equations are numerically solved for every particle in the system within an instance of time making MD computationally costly. Using modern concepts like parallelization MD has gained significant performance improvements in recent years. But, still, these exhaustive simulations have become a bottleneck in the majority of biological processes. MD has a number of applications in thermodynamic, bioinformatics, material science, chemistry, simulations of protein shapes, and refinement of X-ray structures. Implementations of MD using FPGAs and OpenCL takes considerable design time and skills. Interactions of atoms mainly fall into two categories called Bonded and Non-bonded interaction. The former ones are only affected by few neighboring atoms since they are linked by covalent bonds and the latter ones have interactions with many neighbors making MD computationally intensive.

In [34], they propose an OpenCL based heterogeneous computing system with an FPGA accelerator. They have implemented the most time consuming non-bonded interaction computations using the FPGA accelerator. Since the atoms move with time, the number of atoms in a cell is not a constant making the loop boundaries data-dependent and not suitable for OpenCL implementation. To get around this issue, they introduced a pipelined architecture replacing nested loops.

The evaluation (Table XII, XIII) of the implementation is done using DE5a-NET Arria 10 FPGA Development Kit which contains an Arria 10 FPGA and Quartus 16.1 with OpenCL SDK. The heterogeneous system consists of an Intel Xeon E5 CPU with 64GB RAM with CentOS 7. They simulated a box of 22,795 atoms in a box in a dimension of 61.24 x 10-10 m3.

Implementation	Processing Time(s)
CPU	0.68
FPGA(using the same CPU code)	88.03
FPGA(using the proposed method)	0.17

TABLE XII: Comparison of the non-bonded force computation time on CPU and FPGA

Task	Processing Time(s)
Non-bonded Force Computation in FPGA	0.14
Bounded Force Computation in CPU	0.14
Data Trasfer: CPU to FPGA	0.25
Data Trasfer: FPGA to CPU	0.08
Total Processing Time(parallel processing on CPU and FPGA)	0.47
Total Processing Time in Conventional Method(using CPU only)	0.82

TABLE XIII: Processing time for different tasks in the heterogeneous system per one ite ration

In [35], they tried to experiment and determine whether the OpenCL implementation is competitive with an HDL implementation of MD using several designs with pipelines: single-level implementations in Verilog and OpenCL, a two-level Verilog implementation with the optimized arbiter and several two-level OpenCL implementations with different arbitration and hand-shaking mechanisms including one with an embedded Verilog module.

They use a method of dividing the cell into 12 slices to remedy the load imbalance (unlikely to process a similar number of particles within a cut-off by each filter kernel). Filter kernels take pairs of particles and output details of ones within the cut-off radius using channels. They propose seven different designs making a decision when data is available in multiple channels named No filter, Round robin, Explicit handshaking, Explicit handshaking with custom logic, Distributed control, Verilog without, and with filtering.

The evaluation is done on a Nallatech 385A FPGA card, which hosts an Altera Arria 10 FPGA and has an 8GB DDR3 on-card memory, and Altera OpenCL SDK 16.0 support. The summary of results obtained by performing different experiments are shown in Table XIV

Design	LEs(%)	FFs(%)	BRAMs(%)	DSPs(%)	Frq(MHz)	Time(ms)
1 OpenCL No Filter	9 1	15 1	11 3	89 1	151.3	3.9
2 OpenCL Round Robin	13 5	21 7	27 18	9 5	246.1	32.0
3 OpenCL Standard	13 5	20 6	27 17	9 5	245.2	22.1
4 OpenCL Hybrid	19 10	31 17	43 33	9 5	241.1	17.0
5 OpenCL Distrib.	14 5	24 9	29 17	8 4	256.5	3.1
6 Verilog No Filter	1	1	2	2	194.8	3.0
5 Verilog Standard	1	1	2	6	195.2	0.5

TABLE XIV: Collected results

IV. Related Work Results Summary

Following are the key takeaway points in terms of optimizing FPGA based accelerations using OpenCL.

- Larger pipelines lead to better performance but at the cost of higher resource consumption.
- The use of smaller data types for kernel code results in better performance and less resource consumption on FPGAs.
- Data level parallelism is important to achieve successful performance rates at the expense of a moderate increase in resource usage.
- When considering DNA sequencing algorithms, larger workloads benefit all kernels regardless of sequence similarity.
- When considering power efficiency, most of the FPGA accelerators are better than GPU based implementations.
- The exploitation of OpenCL memory hierarchy, such as the private memory offers considerable benefits, although constant memory usage hardly improves the performance.
- Data transfer time between CPU and FPGA is a performance bottleneck. This can be eliminated using unified memory space for CPU and FPGA.
- Since OpenCL allows multiple devices exploitation, the workload can be distributed among multiple FPGAs to achieve better performance.
- Unlike the existing HDL-based alternatives, OpenCL paradigm facilitates portability.

Following tables show a summary of results obtained by different implementations of SW, KNN and CNN algorithms based on common criteria that are available on research articles we reviewed.

Research	Rucci et al.	Sirasao et al.	Benkrid et al.
FPGA	Altera Stratix V	Xilinx Virtex-7 690T	XILINX XC2V6000-4
PEs	-	32	100
GCUPS	37.64–113.78	77.00	6.67
Watts(W)	25	28.00	-
GCUPS/Watts	1.51–0.13	2.8	-

TABLE XV: Performance comparison between different SW implementations

Research	Muslim et al. (impmentation I)	Muslim et al. (impmentation II)
FPGA	ADM-PCIE-7V3 FPGA	ADM-PCIE-7V3 FPGA
Total Execution Time/ms	5.24	1.23
Power(Device)/W	0.422	3.136
Energy(Device)/mJ	0.523	3.9

TABLE XVI: Performance comparison between different KNN implementations

Research	Suda et al.	Zhang et al.	
FPGA	Stratix-V GSD8	Arria 10 GX1150	
Frequency(MHz)	120	370	385
CNN Size(GOP)	30.9	30.76	
Precision	Fixed	Float	Fixed
Throughput	136.5	866	1790

TABLE XVII: Performance comparison between implementations of VGG CNN

** GOPS - Giga [billion] Operations Per Second*

V. Summary

This paper reviewed previous work done to optimize high-performance computing applications using OpenCL on FPGAs to achieve better performance relative to state-of-art implementations while minimizing energy consumption. CPU/GPU implementations usually show better performance due to their novel technology usage. But according to the literature, they consume a more power relative to FPGAs making them not applicable for portable embedded systems. Several areas including bioinformatics, molecular dynamics, machine learning, deep learning, and other computationally intensive applications such as stencil computations are selected for review. Under bioinformatics, we selected one of the widely used Smith-Waterman algorithms and reviewed different approaches and their impact. Molecular dynamics is an area where the physical behavior of atoms or molecules is analyzed using simulations. Each iteration of such simulation has to calculate Newtonian equations per huge number of the particle being real-time at the same time. We reviewed what researchers have introduced as alternative methods to overcome the overhead of traditional loops using the flexibility of OpenCL on FPGAs. Machine learning is the most popular discipline in Artificial Intelligence and one of the computationally intensive algorithms in Machine Learning is Nearest Neighbor where all the vertices have to be considered in each iteration. It has many applications such as data clustering and pattern recognition. Convolutional Neural Networks has a major role in computer vision and voice recognition. Due to the huge number of inputs per convolution layer, the number of convolution layers, and floating-point storage and calculations CNN is challenging to run on an embedded device. Several researchers propose solutions such as fixed-point numbers to overcome those problems. Stencil computations are a grid-based iterative computational method with a large number of parallel operations. Researchers have taken the advantage of the arrangement of stencils to achieve better performance.

References

[1] H. M. Waidyasooriya, M. Hariyama, and K. Uchiyama, Design of FPGA-based computing systems with openCL. 2017.

[2] A.V.D Ploeg, *Why use an FPGA instead of a CPU or GPU*, Medium, August 14, 2018. Accessed on: November 11, 2020. [Online]. Available: https://blog.esciencecenter.nl/why-use-an-fpga-instead-of-a-cpu-or-gpu-b234cd4f309c

[3] M. Psarakis, A. Dounis, A. Almabrok, S. Stavrinidis, and G. Gkekas, “An FPGA-Based Accelerated Optimization Algorithm for Real-Time Applications,” J. Signal Process. Syst., vol. 92, no. 10, pp. 1155–1176, 2020, doi: 10.1007/s11265-020-01522-5.

[4] H. M. Hussain, K. Benkrid, and H. Seker, “The role of FPGAs as high performance computing solution to bioinformatics and computational biology data,” Int. Conf. Appl. Informatics Heal. Life Sci., no. November 2014, pp. 102–105, 2013, doi: 10.13140/2.1.3830.7529.

[5] M. I. Alali, K. M. Mhaidat, and I. A. Aljarrah, “Implementing image processing algorithms in FPGA hardware,” 2013 IEEE Jordan Conf. Appl. Electr. Eng. Comput. Technol. AEECT 2013, 2013, doi: 10.1109/AEECT.2013.6716446.

[6] R. J. Petersen and B. L. Hutchings, “An assessment of the suitability of FPGA-based systems for use in digital signal processing,” Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), vol. 975, pp. 293–302, 1995, doi: 10.1007/3-540-60294-1_123.

[7] S. Qasim, S. Abbasi, and A. Bandar, "Advanced FPGA Architectures for Efficient Implementation of Computation Intensive Algorithms: A State-of-the-Art Review," MASAUM J. Comput., vol. 1, no. 2, pp. 300–303, 2009.

[8] K. Nagarajan, B. Holland, A. D. George, K. C. Slatton, and H. Lam, "Accelerating machine-learning algorithms on FPGAs using pattern-based decomposition," J. Signal Process. Syst., vol. 62, no. 1, pp. 43–63, 2011, doi: 10.1007/s11265-008-0337-9.

[9] M. Leeser, S. Coric, E. Miller, H. Yu, and M. Trepanier, "Parallel-beam backprojection: An FPGA implementation optimized for medical imaging," J. VLSI Signal Process. Syst. Signal Image. Video Technol., vol. 39, no. 3, pp. 295–311, 2005, doi: 10.1007/s11265-005-4846-5.

[10] H. Guo, L. Su, Y. Wang, and Z. Long, "FPGA-accelerated molecular dynamics simulations system," Int. Conf. Scalable Comput. Commun. - 8th Int. Conf. Embed. Comput. ScalCom-EmbeddedCom 2009, pp. 360–365, 2009, doi: 10.1109/EmbeddedCom-ScalCom.2009.71.

[11] N. Street, "VERILOG HDL based FPGA design Gary Gannot and Michiel Ligthart Exemplar Logic , Inc .," 1994.

[12] C. Rust, F. Stappert, R. Künnemeyer, R. Kuennemeyer, J. Cong, and B. Liu, "From Timed Petri Nets to to Interrupt-Driven Embedded Control Software," … -Aided Des. …, vol. 30, no. 4, pp. 473–491, 2003.

[13] T. S. Czajkowski et al., "From OpenCL to high-performance hardware on FPGAs," Proc. - 22nd Int. Conf. F. Program. Log. Appl. FPL 2012, no. March, pp. 531–534, 2012, doi: 10.1109/FPL.2012.6339272.

[14] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "OpenCL-based FPGA-platform for stencil computation and its optimization methodology," IEEE Trans. Parallel Distrib. Syst., vol. 28, no. 5, pp. 1390–1402, 2017, doi: 10.1109/TPDS.2016.2614981.

[15] B. P. Guide, "Intel® FPGA SDK for OpenCLTM," pp. 1–122, 2016.

[16] M. J. Reigosa et al., "Comparison of physiological effects of allelochemicals and commercial herbicides," Allelopath. J., vol. 8, no. 2, pp. 211–220, 2001.

[17] E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matias, "Accelerating smith-waterman alignment of long DNA sequences with OpenCL on FPGA," in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 2017, vol. 10209 LNCS, pp. 500–511, doi: 10.1007/978-3-319-56154-7_45.

[18] E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matias, "SWIFOLD: Smith-Waterman implementation on FPGA with OpenCL for long DNA sequences," BMC Syst. Biol., vol. 12, no. Suppl 5, 2018, doi: 10.1186/s12918-018-0614-6.

[19] E. Rucci, C. Garcia, G. Botella, A. De Giusti, M. Naiouf, and M. Prieto-Matias, "Smith-Waterman Protein Search with OpenCL on an FPGA," Proc. - 14th IEEE Int. Conf. Trust. Secur. Priv. Comput. Commun. Trust. 2015, vol. 3, pp. 208–213, 2015, doi: 10.1109/Trustcom.2015.634.

[20] A. Sirasao, E. Delaye, R. Sunkavalli, and S. Neuendorffer, "Fpga based opencl acceleration of genome sequencing software," System, vol. 128, no. 8.7, p. 11, 2015.

[21] K. Benkrid, Y. Liu, and A. Benkrid, "A highly parameterized and efficient FPGA-Based skeleton for pairwise biological sequence alignment," IEEE Trans. Very Large Scale Integr. Syst., vol. 17, no. 4, pp. 561–570, 2009, doi: 10.1109/TVLSI.2008.2005314.

[22] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "OpenCL-based FPGA-platform for stencil computation and its optimization methodology," IEEE Trans. Parallel Distrib. Syst., vol. 28, no. 5, pp. 1390–1402, May 2017, doi: 10.1109/TPDS.2016.2614981.

[23] S. Wang and Y. Liang, "A Comprehensive Framework for Synthesizing Stencil Algorithms on FPGAs using OpenCL Model," Proc. - Des. Autom. Conf., vol. Part 12828, no. c, 2017, doi: 10.1145/3061639.3062185.

[24] Q. Jia and H. Zhou, "Tuning Stencil codes in OpenCL for FPGAs," Proc. 34th IEEE Int. Conf. Comput. Des. ICCD 2016, pp. 249–256, 2016, doi: 10.1109/ICCD.2016.7753287.

[25] Z. Zhang, "Introduction to machine learning: K-nearest neighbors," Ann. Transl. Med., vol. 4, no. 11, 2016, doi: 10.21037/atm.2016.03.37.

[26] Y. Pu, J. Peng, L. Huang, and J. Chen, "An efficient KNN algorithm implemented on FPGA based heterogeneous computing system using OpenCL," Proc. - 2015 IEEE 23rd Annu. Int. Symp. Field-Programmable Cust. Comput. Mach. FCCM 2015, pp. 167–170, 2015, doi: 10.1109/FCCM.2015.7.

[27] F. Muslim, A. Demian, L. Ma, L. Lavagno, and A. Qamar, "Energy-efficient FPGA Implementation of the k-Nearest Neighbors Algorithm Using OpenCL," Position Pap. 2016 Fed. Conf. Comput. Sci. Inf. Syst., vol. 9, no. October 2020, pp. 141–145, 2016, doi: 10.15439/2016f327.

[28] U. Guide, "SDAccel Development Environment," vol. 1023, pp. 1–79, 2015.

[29] N. Suda et al., "Throughput-optimized openCL-based FPGA accelerator for large-scale convolutional neural networks," FPGA 2016 - Proc. 2016 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp. 16–25, 2016, doi: 10.1145/2847263.2847276.

[30] T. F. Gonzalez, "Handbook of approximation algorithms and metaheuristics," Handb. Approx. Algorithms Metaheuristics, pp. 1–1432, 2007, doi: 10.1201/9781420010749.

[31] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," 3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc., pp. 1–14, 2015.

[32] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," FPGA 2017 - Proc. 2017 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays, pp. 25–34, 2017, doi: 10.1145/3020078.3021698.

[33] M. Alam, P. Bethoju, and M. Song, "Study of traffic engineering capabilities of MPLS networks," Int. Conf. Inf. Technol. Coding Comput. ITCC, vol. 2, pp. 14–15, 2005, doi: 10.1109/itcc.2005.264.

[34] H. M. Waidyasooriya, "OpenCL-Based Implementation of an FPGA Accelerator for Molecular Dynamics Simulation," vol. 3, no. 2, pp. 11–23, 2017.

[35] C. Yang, J. Sheng, R. Patel, A. Sanaullah, V. Sachdeva, and M. C. Herbordt, "OpenCL for HPC with FPGAs: Case study in molecular electrostatics," 2017 IEEE High Perform. Extrem. Comput. Conf. HPEC 2017, 2017, doi: 10.1109/HPEC.2017.8091078.