Accelerating Adaptive Banded Event Alignment Algorithm Using OpenCL

- Semester 7 Report -

Wishma Herath
Suneth Samarasinghe
Pubudu Premathilaka

Department of Computer Engineering
University of Peradeniya

CO421 : Final Year Project (course CO421) report submitted as a requirement of the degree of
B.Sc.Eng. in Computer Engineering

October 2019

Supervisors: Prof. Roshan Ragel(University of Peradeniya) and Mr. Hasindu Gamaarachchi(University of New South Wales)

I would like to dedicate this thesis to my loving parents and “teachers” . . .

Declaration

I/We hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is our own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgments.

Wishma Herath
Suneth Samarasinghe
Pubudu Premathilaka
October 2019

Acknowledgements

We would like to thank the immense support and the guidance given by our project supervisors Mr. Hasindu Gamaarachchi and Prof. Roshan Ragel. A very special gratitude goes to the Faculty for providing necessary equipment and support for this project.

Abstract

Nanopore sequencing is one of the most promising technologies being developed as a cheap and fast alternative to conventional sequencing methods, especially in the sequencing of polynucleotides in the form of DNA or RNA. Applications involved in the field of direct contact with genotyping and point-of-care diagnostics require efficient bioinformatic algorithms for the analysis of raw nanopore signal data.

To perform these requirements efficiently, the utilization of an optimized bioinformatics algorithm causes a significant change in the field of nanopore sequencing. Adaptive Banded Event Alignment(ABEA) a one such commonly used algorithm and the original implementation of ABEA in the Nanopolish software package has already been parallelized and optimized for GPUs(named as f5c). It performs efficiently on heterogeneous CPU-GPU architectures.

Even though the ABEA algorithm has been fine-tuned, to exploit architectural features in GPUs, the hardware on such generic processors cannot be modified. It will be a tremendous achievement if customized hardware performs with the optimized ABEA algorithm concurrently aiming for more performance improvement of the overall system. In this project, we deploy an optimized version of the ABEA algorithm on a FPGA, using OpenCL to achieve better performance.

Table of contents

List of figures

List of tables

Nomenclature

Acronyms / Abbreviations

API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
CU	Compute Unit
CUDA	Compute Unified Device Architecture
DNA	Deoxyribonucleic Acid
DP	Dynamic Programming
FIR	Finite impulse response
RNA	Ribonucleic Acid
RTL	Register Tansfer Level
SW	Smith-Waterman
VHDL	Register Transfer Level

Chapter 1

Introduction

In the field of biology, it has been facing a rapid development in data from many types of research in the same way as other scientific directions. Nowadays, vast volumes of data are generated related to biomedical fields in sequencing centres, analytical facilities, individual laboratories, etc. Thus, obtaining the relevant information within a certain period of time is a challenging task that is faced by the scientific community.

Genomic medicine is a developing medical discipline that incorporates using genomic information. In applications like clinical diagnosis, rapid species identification, and advanced therapies, genomics plays a huge contribution to the biomedical research field. Genomic medicine includes early diagnosis, more efficient disease prevention and management, and the reduction of medication side-effects dependent on gene signatures. Therefore, new methods of developing low-cost full genome DNA sequences play a huge role in the future of genomics.

Nowadays, the focus of genome projects has moved from data production to data analysis, and also the central challenge is how to analyze such an amount of data rapidly and accurately. The modern sequencing methods generate data such that traditional analysis tools are not able to cope with them. Therefore, DNA analysis algorithms have to be implemented on hardware accelerators to achieve the expectations efficiently and effectively.

1.1 Background

1.1.1 Nanopore Sequencing

Nanopore sequencing is a special technique for direct, real-time analysis of long DNA or RNA fragments. It operates by measuring changes in electrical current when nucleic acids move through the nanopore protein. The resulting signal is decoded to produce a particular DNA or RNA sequence. When considering the overall cost of nanopore sequencing, in 2012 the cheapest cost is about \$18k per genome including usage and analysis costs from a high-throughput sequencing centre.

To reduce that amount of cost(i.e. instrument and usage costs), nanopores are being pursued as next-generation sequencing (NGS) platforms. From NGS, it leads to getting rid of the requirement of sample amplification, the use of enzymes and reagents used for catalytic function during the sequencing operation, and optics for detection of sequencing progress.

When considering the latest generation of sequencing technologies, which is the 3rd generation, able to build ultra-long DNA reads of a molecule in real-time. Especially using MinlON, i.e: a pocket-size sequencing device manufactured by Oxford Nanopore Technologies qualified for sequencing at hospitals, clinics, etc[1].

1.1.2 Methylation Calling

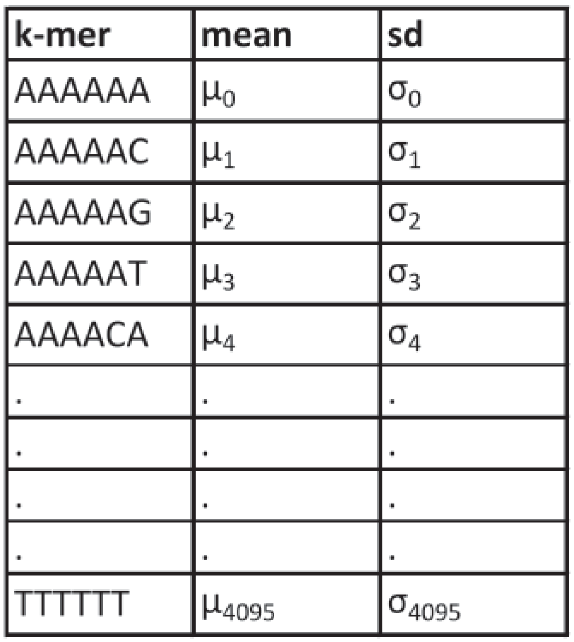
Nanopore sequencing offers real-time analysis, at the expense of a higher error rate, which is predominantly caused by the conversion of the raw signal into DNA bases via probabilistic models and it is referred to as ‘base-calling’. To overcome base-calling errors, the raw signal can be revisited as a posteriori.

The raw signal can be re-examined as a posteriori to resolve base-calling errors. Such polishing could be accurate by aligning the raw signal with a biological reference sequence for base-calling errors and thus detect raw signal idiosyncrasies by comparing observed signal rates with predicted levels at all associated positions[2].

As mentioned earlier base calling causes essential biological information to be lost. Some base-calling models cannot handle methylated data either because they are trained in non-methylated sequences, or because non-canonical bases are abstracted. Such molecules can also be wrongly marked as non-methylated bases. The methylation calling means the process of identifying methylation[2].
Three steps are to be followed for a given read under methylation calling and steps are performed for each reading in the data set,

1. Event detection
2. Signal-space alignment
3. Hidden Markov Model (HMM) profiling

Event detection is the time series segmentation of the raw signal based on sudden signal level changes. Each of these segment is called *event* and it is denoted by mean($\mu_{\overline{x}}$), standard deviation ($\sigma_{\overline{x}}$) and the duration of the raw signal samples ($n_{\overline{x}}$). To obtain a better match between events and the raw signal, events are aligned to a generic k-mer model signal. Nanopolish software package accomplish this task using Adaptive Banded Event Alignment(ABEA) algorithm. The alignment between the events and the k-mers in a reference genome can be subjected to Hidden Markov Model (HMM) profiling to identify if a given base is methylated or not[1].



k-mer	mean	sd
AAAAAA	μ_0	σ_0
AAAAAC	μ_1	σ_1
AAAAAG	μ_2	σ_2
AAAAAT	μ_3	σ_3
AAAACA	μ_4	σ_4
.	.	.
.	.	.
.	.	.
.	.	.
TTTTTT	μ_{4095}	σ_{4095}

Fig. 1.1 An Example of k-mer Model

1.1.3 Adaptive Banded Event Alignment Algorithm (ABEA)

In 2017, a heuristic algorithm Suzuki-Kasahara (SK)[3] was implemented to increase the processing speed. SK uses an adaptive band scheme that allows a shorter band to accommodate such kind of alignment. However, the band is no longer enough for the whole alignment to lead to an unsatisfactory alignment. It overcomes with the use of an adaptive unit. Redesigned versions of the SK algorithm is used in Nanopolish for event-space alignment and it is called ABEA.

The pseudo-code of ABEA algorithm presented in nanopolish software package is shown in Figure 1.2. It takes the sequenced read in base-space(*ref*), the output of the event detection step(*events*) and the pore-model(*model*) as inputs and outputs the alignment as *event-ref* pairs.

First, it initializes the first two bands[line 2], then the outer loop iterates through the rest of the bands from top-left to bottom-right of the DP table and the inner loops[line 11-15] iterates through the cells inside the band. Line 4-9 corresponds to handling the band movement on the DP table. Band movement handles form line 4-9. An alignment score is calculated for each cell using a specific cost function with heuristic-based constants. Finally, after all the cells on the DP table are calculated, actual alignment is found using backtracking operation.

1.1.4 OpenCL for FPGA Architecture and Programming

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.

The most common approach to achieve better performance is by assigning the computationally intensive task to hardware and exploiting the parallelism in the algorithm[4]. 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 on the spectrum of processing elements[5].

Historically, hardware developers used hardware description languages (HDL), also known as High-level programming environments like Verilog and VHDL to program hardware at register transfer level (RTL)[6]. 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.

This forced developers to come up with high-level synthesis (HLS) tools like Intel OpenCL (Open Computing Language) HLS and Xilinx Vivado HLS[7]. 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

Input:
ref[]: the base-called read (1D char array)
events[]: event table containing $\{\mu_{\overline{x}}, \sigma_{\overline{x}}, n_{\overline{x}}\}$ *of each event—1D {float,float,float} array*
model: pore-model
Output:
alignment[]: alignment denoted by a list of {event index,k-mer index}—1D {float,float,float} array

```
function ALIGN($ref, model, events$))
    $initialise\_first\_two\_bands(score, trace, ll\_idx)$
    for $i \leftarrow 2$ to n_bands do
        $idir \leftarrow suzuki\_kasahara\_rule(score[i-1])$
        if $dir == right$ then
            $ll\_idx[i] \leftarrow move\_band\_to\_right(ll\_idx[i-1])$
        else
            $ll\_idx[i] \leftarrow move\_band\_down(ll\_idx[i-1])$
        end if
        $min\_j, max\_j \leftarrow get\_limits\_in\_band(ll\_idx[i])$
        for $j \leftarrow$ min_j to max_j do
            $s, d \leftarrow compute(score[i-1], score[i-2], ref, events, model)$
            $score[i, j] \leftarrow s$
            $trace[i, j] \leftarrow d$
        end for
    end for
    $alignment \leftarrow backtrack(score, trace.ll)$
end function
```

Fig. 1.2 Adaptive Banded Event Alignment(ABEA)

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.

In[8], 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[9]. The programs written in OpenCL can then be converted to RTL designs to support a wide variety of platforms. An OpenCL platform comprises one host and one or many devices, which are computed units that may consist of multiple processing elements (PEs).

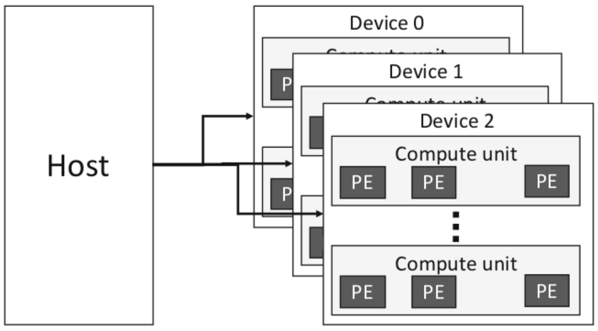


Fig. 1.3 OpenCL Platform Model

OpenCL platform model[10], an abstract hardware model for devices (Figure 1.3). 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).

The host program performs the following tasks

- 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.

Figure 1.4 illustrates the schematic diagram of the Intel FPGA SDK for the OpenCL programming model.

The execution model (Figure 1.5) 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.

OpenCL for FPGA uses two types of kernels, namely 'Single work item' kernels and 'NDRange kernels'[10] (Figure 1.6). 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.

In[10], 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.

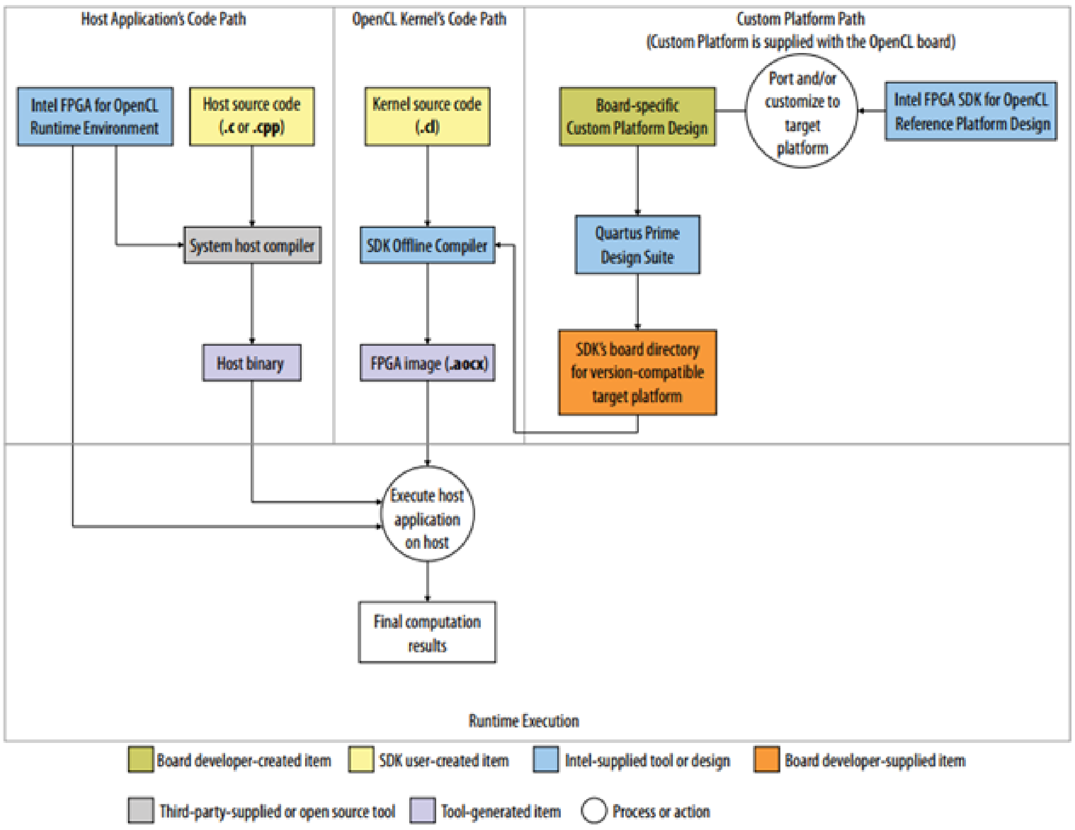

Host Application's Code Path
OpenCL Kernel's Code Path
Custom Platform Path
(Custom Platform is supplied with the OpenCL board)
Intel FPGA for OpenCL Runtime Environment
Host source code (.c or .cpp)
Kernel source code (.cl)
Board-specific Custom Platform Design
Port and/or customize to target platform
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. 1.4 Schematic diagram of the Intel FPGA SDK for OpenCL programming model

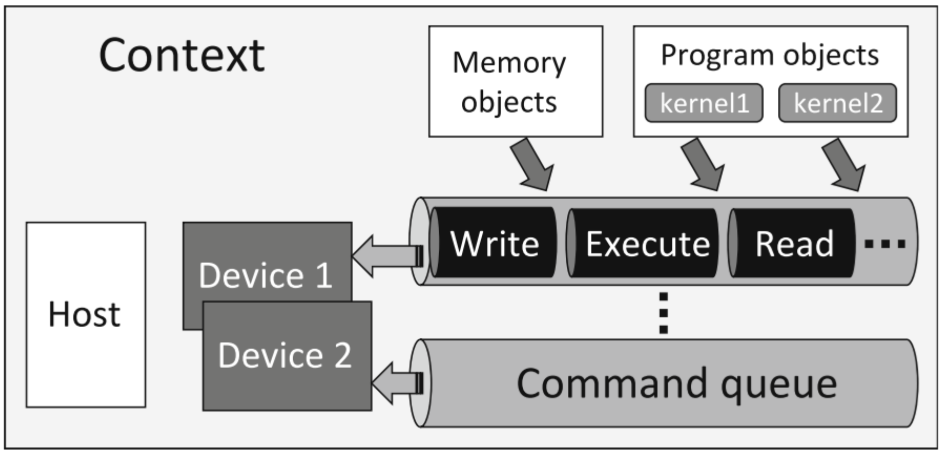


Fig. 1.5 OpenCL Execution Model

The memory hierarchy of OpenCL is shown in Figure 1.7. 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

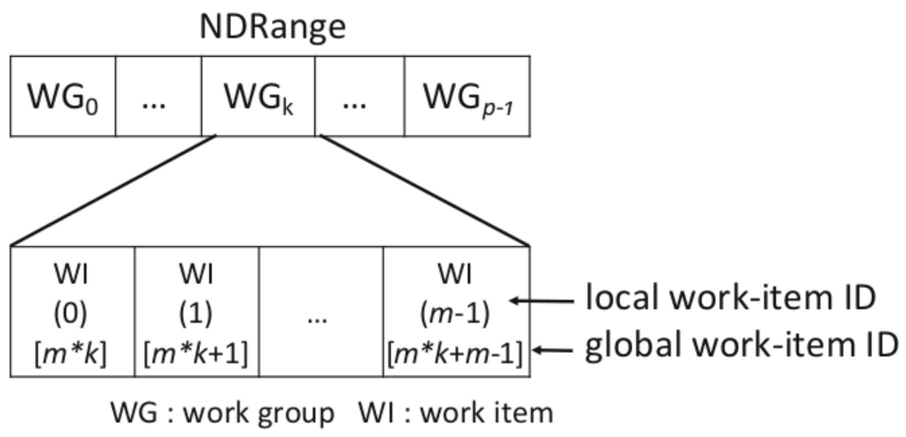


Fig. 1.6 OpenCL Kernel Programming Model

group has a local memory shared by each work item and a work item has its own private memory.

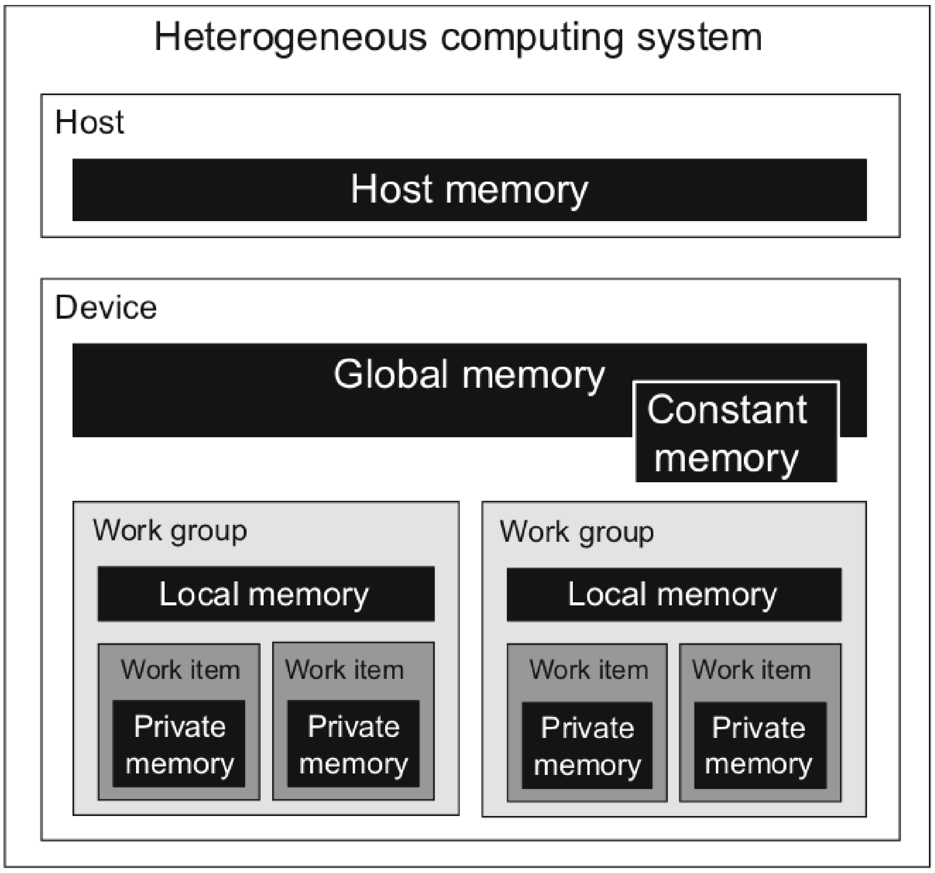


Fig. 1.7 OpenCL Memory Model

1.2 Problem Statement

The dynamic programming algorithm ABEA is a time-consuming step in nanopolish software packages that come under nanopore DNA sequencing. It has been discovered that it consumes ≈70% of the total CPU time in execution. Therefore, it is required to investigate strategies to reduce the runtime of ABEA for nanopore applications. GPU implementation does not have the capability of automatic parameter tuning at run-time. Also, it has achieved 3-5x performance increase compared to CPU version.

1.3 Proposed Solution

We propose to address the limitations of the latest GPU version of ABEA. Using FPGA based implementation, we hope to achieve better performance and power utilization. Using OpenCL, we develop a portable version of the algorithm which can deploy in FPGAs, GPUs, CPUs and other processors and accelerators.

Chapter 2

Related Work

2.1 GPU Accelerated Adaptive Banded Event Alignment Algorithm

Previous research[1], which is done under the objective of accelerating ABEA, deployed an accelerated version of the algorithm on GPUs using CUDA. In this work, high read length variability was one of the key problems that were solved by a variety of memory optimizations and a heterogeneous computing approach that uses both CPU and GPU. They have achieved 3-5x performance improvement on the CPU-GPU system when compared to the original CPU version of the nanopolish software package. As of now The complete methylation calling of a human genome can be performed in real-time while the nanopore sequencer is operating on an embedded system such as in an SoC equipped with an ARM processor and an NVIDIA GPU. They have re-engineered the original Nanopolish methylation detection tool to efficiently utilize existing CPU resources, which they have referred to as f5c. According to its results, f5c powered by GPU accelerated ABEA can process the output from the rest of the pipeline on a single NVIDIA TX2 SoC, at a speed of (>600 Kbases per second) to keep up with the sequencing output as shown in Figure 2.1.

Also they have shown that, if the original Nanopolish was executed on the NVIDIA TX2 SoC, the processing speed is limited to ≈256 Kbases per second. Their work will not only reduce the associated costs of Nanopore data processing and data transfer but will also improve the turnaround time of the final test outcome.

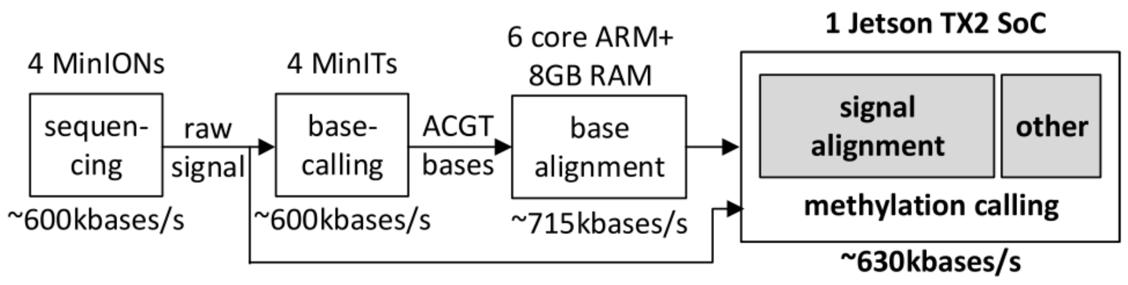


Fig. 2.1 Human Genome Processing on-the-fly

2.2 Algorithms Accelerated on FPGAs

2.2.1 Smith-Waterman Algorithm Acceleration Using OpenCL

Smith-Waterman (SW)[11] algorithm is a widely used pairwise sequence alignment algorithm that finds the best possible aligned sub-segment in a pair of sequences. Accelerating SW is a great challenge in the field of high-performance computation.

In[12], 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 vertical blocks. 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. They showed that using smaller data types for kernel implementation has increased the performance as well as it has reduced resource consumption.

In[13], by Rucci et al. 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.

In[14], Sirasao et al. have presented FPGA and OpenCLbased acceleration to the SW algorithm. They have benchmarked performance per watt on different hardware platforms including CPUs, GPUs, and FPGAs. Also, it presents a performance tradeoff using OpenCL based programming environment.

2.2.2 High-Performance Stencil Computation 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.

In 2017, Waidyasooriya et al. propose an FPGA platform using OpenCL for stencil computations[15] 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[15]. They have achieved higher processing speed relative to multicore CPU and GPU implementations and more than 60% of the peak performance given by FPGA.

In[16], Wang et al. 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.

In[17], 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.

2.2.3 K-Nearest Neighbor Algorithm (KNN) using OpenCL

K-Nearest Neighbor Algorithm(KNN) is one of the most popular machine learning algorithms[18] and due to high computational complexity for large datasets, it has become popular in the field of high-performance computing.

In[19], Pu et al. have proposed a new solution to speed up the KNN algorithm on FPGA based heterogeneous computing systems with OpenCL. They have introduced a specific bubble sorting algorithm based on FPGA's parallel pipeline structure to optimize the KNN algorithm. The GPU accelerated our KNN algorithm by 410 times the speed of the 4-threads CPU implementation, while FPGA achieved 148 times. 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[20] by Muslim et al. Furthermore, they have compared the performance between GPU and FPGA implementations of the same algorithm. In the first approach, both sorting and nearest neighbor identification are performed by the host. It uses only global memory and because the independent data usage algorithm is 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. 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.

2.2.4 Convolutional Neural Networks (CNN) using OpenCL

It is challenging to apply CNNs 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[21], 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[22] and VGG on FPGAs.

In[23], 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. 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[24], 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 of OpenCL kernels (namely Convolutional kernel, Data mover kernel, and other kernels) integrated using Altera's OpenCL extension channels. Throughput optimization is done by data vectorization and parallelization of CUs. Optimizations of bandwidth are achieved by introducing a sliding-window data buffering scheme and fixed-point arithmetic is used instead of floating-point to reduce memory bandwidth requirements and hardware costs.

2.2.5 Molecular Dynamics Applications using OpenCL

Molecular dynamic (MD) is the area of computer simulations to analyze the physical behavior of atoms and molecules in space. The simulation is driven by the numerical results given by relatively applying classical Newtonian dynamic equations to atoms or molecules.

In[25], 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.

In[26], 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.

2.2.6 Takeaways from Related Work

Following are the key 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.

Chapter 3

Design and Implementation

In the first phase of our project, we have followed the procedure described in[1]. The ultimate goal of both of these approaches is to optimize the ABEA algorithm. Following, we describe the methodologies we used based on[1]. In the next phase, we fine-tune the implementation based on FPGA specific optimization techniques.

In this system, the application starts out executing on the CPU, and then the CPU launches kernels on FPGA. The data transferred between host and the device is done using the PCIe bus to minimize the impact of communication.

3.1 Optimization Techniques

The original CPU implementation processes a read at a time. But, in OpenCL implementation, a batch of reads is processed at a time to optimize the performance by data transfer overhead between the host main memory and device memory and by allowing the maximum usage of device resources for parallelization.

3.1.1 Decomposition of the Algorithm into Multiple Kernels

All-in-one kernels tend to use a large number of registers. But, typical OpenCL devices only have a finite number of register file sizes. Therefore, fewer concurrent warps, large kernels often result in poor overall performance compared to multiple kernels. The GPU implementation of Adaptive Banded Event Alignment (ABEA) algorithm[1] consists of three kernels since multiple kernels allow efficient thread assignments.

- Pre kernel: Initialising the first two bands of the dynamic programming table and pre-computing frequently accessed values by the next kernel.

- Core kernel: The filling of the dynamic programming table which is the compute intensive portion of the ABEA algorithm.
- Post kernel: Performs backtracking.

3.1.2 Memory Optimization Techniques

Dynamic memory allocation performed inside kernels are extraordinarily expensive[1]. Therefore dynamic allocation of memory per millions of reads is an extreme overhead and leads to poor performance. In this implementation, a different methodology is used to reduce the number of memory allocations by pre-allocating large chunks of contiguous memory at the beginning of the program to accommodate a batch of reads, and then they are reused throughout the program.

3.2 Implementation

The isolated alignment algorithm is implemented in OpenCL and tested by comparing the output with dumped input and corresponding output of CPU implementation of the algorithm.

The process flow of the testbed is illustrated in Figure 3.1.

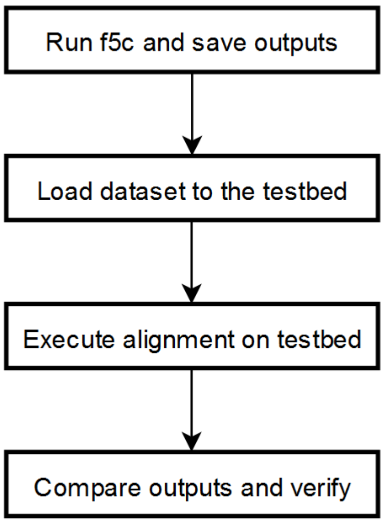

Run f5c and save outputs
Load dataset to the testbed
Execute alignment on testbed
Compare outputs and verify

Fig. 3.1 Process Flow of the Testbed

3.2.1 Experimental Setup

Platform Information (Host)

The host is a low-end personal computer with hardware specifications in Table 3.1. It acts as both the host and the FPGA emulator device.

CL_PLATFORM_NAME	Intel(R) FPGA SDK for OpenCL(TM)
CL_PLATFORM_VERSION	OpenCL 1.0 Intel(R) FPGA SDK for OpenCL(TM), Version 18.0
CPU	Intel Core i5-4200H 2.80GHz x 2
RAM(GB)	12

Table 3.1 Host Specifications

Device Information (Emulator)

Intel FPGA emulation platform (Specifications shown in Table 3.2) for OpenCL is used to assess the functionality of kernels. It is capable of providing rapid compilation time, source code portability between FPGA emulator and physical FPGA hardware. It also give reasonable performance with an average benchmark run at 5x to 10x slowdown in comparison to physical FPGA hardware.

CL_DEVICE_NAME	EmulatorDevice : Emulated Device
CL_DEVICE_VERSION	OpenCL 1.0 Intel(R) FPGA SDK for OpenCL(TM), Version 18.0
CL_DEVICE_ADDRESS_BITS	64
CL_DEVICE_GLOBAL_MEM_SIZE	12473344000B
CL_DEVICE_MAX_CLOCK_FREQUENCY	1000MHz
CL_DEVICE_MAX_COMPUTE_UNITS	1
CL_DEVICE_MAX_CONSTANT_ARGS	8
CL_DEVICE_MAX_WORK_ITEM_DIMENSIONS	3

Table 3.2 Emulator Specifications

3.2.2 Dataset

The dataset we use for experiments on OpenCL implementation of ABEA algorithm is a subset of publicly available reads that aligned to a 2kb region in the E. coli draft assembly (Table 3.3). Statistical information of the testset is in Table 3.4.

Sample	E. coli str. K-12 substr. MG1655
Instrument	MinION sequencing R9.4 chemistry
Basecaller	Albacore v2.0.1
Region	”tig00000001:200000-202000”
Note	Ligation-mediated PCR amplification performed

Table 3.3 Details of the Dataset

Dataset	Number of reads	Number of bases	Mean read length (Bases)	Max read length (Bases)
testset	143	819102	5727	12618

Table 3.4 Statistical Information of the Dataset

Chapter 4

Results and Analysis

4.1 Results

Execution time for each kernel are measured using gettimeofday function from sys/time.h.

Kernel	Execution time(s)	Percentage
Pre-kernel	0.125	0.008%
Core-kernel	1501.740	99.922%
Post-kernel	1.045	0.070%

Table 4.1 Kernel Execution Times for Testset on OpenCL for DE5net FPGA Emulator

Implementation	Total execution time(s)
FPGA implementation on DE5net FPGA Emulator	1503.126
CPU implementation on Host PC	6.187

Table 4.2 Execution times of CPU and FPGA Emulator implementations

Table 4.3 shows the number of reads that gives the expected alignment pairs as "Passed" and reads that fail to produce the expected alignment pairs as "Failed".

	Reads	Percentage
Passed	109	76.22%
Failed	34	23.77%

Table 4.3 Results Validation of OpenCL Implementation with CPU Implementation

4.2 Analysis

The execution times of each kernel in the OpenCL implementation of the algorithm after running on the Intel FPGA emulator are given in Table 4.1. As we anticipated the Core-kernel takes the longest time of around 99.9% of the total execution time since it is compute intensive.

As shown in Table 3.2, the Intel FPGA emulator has limited resources such as 1GHz maximum clock frequency and one CU compared to a physical FPGA hardware. Eventhough the device global memory size is listed as around 12GB (the total main memory of the host PC), it is only an upper boundary since the PC performs as both the Host and the Emulator Device.

Those limitations of the Intel FPGA emulator justify the significant difference in total execution times of OpenCL on FPGA implementation and original CPU implementation of the algorithm shown in Table 4.2. It is around 240x of slowdown relative the CPU implementation.

Chapter 5

Conclusion and Future Work

The Adaptive Banded Event Alignment algorithm is an improved version of DNA sequencing which is extensively used in nanopore DNA sequencing. The most recent work that has been done by the previous work is that it has parallelized this algorithm and it can efficiently run GPUs and it has achieved 3-5$\times$ performance improvement on the CPU GPU system when compared to CPU.

During the past 10 weeks, we have been able to extract necessary functions (the alignment function) from the original work and convert it into OpenCL such that it works successfully with the FPGA Emulator.

In the next phase, we take this implementation to run on an Altera Stratix V FPGA and try to experimentally identify and adapt FPGA optimization techniques to achieve better performance. Finally, we propose to compare the performance of our implementation with the related work done so far.

References

[1] H. Gamaarachchi, C. W. Lam, G. Jayatilaka, H. Samarakoon, J. Simpson, M. Smith, and S. Parameswaran, "GPU Accelerated Adaptive Banded Event Alignment for Rapid Comparative Nanopore Signal Analysis," pp. 1–37, 2019.

[2] R. D. Maitra, J. Kim, and W. B. Dunbar, "Recent advances in nanopore sequencing," *Electrophoresis*, vol. 33, no. 23, pp. 3418–3428, 2012.

[3] H. Suzuki and M. Kasahara, "Introducing difference recurrence relations for faster semi-global alignment of long sequences," *BMC Bioinformatics*, vol. 19, no. Suppl 1, 2018.

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

[5] K. Nagarajan, B. Holland, A. D. George, K. C. Slatton, and H. Lam, "Accelerating machine-learning algorithms on FPGAs using pattern-based decomposition," *Journal of Signal Processing Systems*, vol. 62, no. 1, pp. 43–63, 2011.

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

[7] 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 Design of . . .* , vol. 30, no. 4, pp. 473–491, 2003.

[8] T. S. Czajkowski, U. Aydonat, D. Denisenko, J. Freeman, M. Kinsner, D. Neto, J. Wong, P. Yiannacouras, and D. P. Singh, "From OpenCL to high-performance hardware on FPGAs," *Proceedings - 22nd International Conference on Field Programmable Logic and Applications, FPL 2012*, no. March, pp. 531–534, 2012.

[9] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "OpenCL-based FPGA-platform for stencil computation and its optimization methodology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, pp. 1390–1402, may 2017.

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

[11] M. J. Reigosa, L. Gonzalez, A. Sanches-Moreiras, B. Duran, D. Puime, D. A. Fernadez, and J. C. Bolano, "Comparison of physiological effects of allelochemicals and commercial herbicides," *Allelopathy Journal*, vol. 8, no. 2, pp. 211–220, 2001.

[12] 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)*, vol. 10209 LNCS, pp. 500–511, Springer Verlag, 2017.

[13] 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 Systems Biology*, vol. 12, no. Suppl 5, 2018.

[14] 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.

[15] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "OpenCL-based FPGA-platform for stencil computation and its optimization methodology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1390–1402, 2017.

[16] S. Wang and Y. Liang, "A Comprehensive Framework for Synthesizing Stencil Algorithms on FPGAs using OpenCL Model," *Proceedings - Design Automation Conference*, vol. Part 12828, no. c, 2017.

[17] Q. Jia and H. Zhou, "Tuning Stencil codes in OpenCL for FPGAs," *Proceedings of the 34th IEEE International Conference on Computer Design, ICCD 2016*, pp. 249–256, 2016.

[18] Z. Zhang, "Introduction to machine learning: K-nearest neighbors," *Annals of Translational Medicine*, vol. 4, no. 11, 2016.

[19] Y. Pu, J. Peng, L. Huang, and J. Chen, "An efficient KNN algorithm implemented on FPGA based heterogeneous computing system using OpenCL," *Proceedings - 2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines, FCCM 2015*, no. July 2015, pp. 167–170, 2015.

[20] F. Muslim, A. Demian, L. Ma, L. Lavagno, and A. Qamar, "Energy-efficient FPGA Implementation of the k-Nearest Neighbors Algorithm Using OpenCL," *Position Papers of the 2016 Federated Conference on Computer Science and Information Systems*, vol. 9, no. October 2020, pp. 141–145, 2016.

[21] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J. S. Seo, and Y. Cao, "Throughput-optimized openCL-based FPGA accelerator for large-scale convolutional neural networks," *FPGA 2016 - Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 16–25, 2016.

[22] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, pp. 1–14, 2015.

[23] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," *FPGA 2017 - Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 25–34, 2017.

[24] D. Wang, J. An, and K. Xu, "PipeCNN: An OpenCL-Based FPGA Accelerator for Large-Scale Convolution Neuron Networks," 2016.

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

[26] 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 Performance Extreme Computing Conference, HPEC 2017*, 2017.