Accelerated HaplotypeCaller DNA Analysis Application on FPGAs Using CAPI

Nikolaos Bampetas

CE-MS-2017-12

Abstract

DNA carries all the information needed to define the genetic structure of an individual. The large amounts of information stored in the DNA makes it costly to store and to process. In this work, a fast and efficient implementation of a Field Programmable Gate Array (FPGA) based, streaming multicore architecture for accelerating variant calling algorithms will be designed. We focused on the HaplotypeCaller which is the variant calling software part of the Genome Analysis Toolkit (GATK), whiich is one of the most widely used DNA analysis tools in the field. The most time consuming part of the HaplotypeCaller is the PairHMM algorithm. PairHMM is a probabilistic algorithm that executes pairwise alignment of two sequences. Starting from an existing single core PairHMM accelerator design, which was implemented using the POWER8 Coherent Accelerator Processor Interface (CAPI), we designed three extra cores of the PairHMM algorithm that can work independently and increase the performance of the overall system. The new accelerator achieves a 2.2x speedup in comparison with the single core. The accelerator is integrated with the HaplotypeCaller and uses CAPI to access a shared processor-accelerator memory for direct communication. A JNI call is used to send the memory addresses to the accelerator, which reduces the communication overhead between the HaplotypeCaller and the accelerator. Results show that the application is not able to saturate the accelerator with data, resulting in accelerator idle time and under-utilization. The accelerator presented idle time for the 26% of the different data sets that were used. It would be beneficial to implement a faster version of the HaplotypeCaller which can load data sets to the accelerator as current data sets are being processed.

Accelerated HaplotypeCaller DNA Analysis Application on FPGAs Using CAPI

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER ENGINEERING

by

Nikolas Bampetas
born in Athens, Greece

Computer Engineering
Department of Electrical Engineering
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology

Accelerated HaplotypeCaller DNA Analysis Application on FPGAs Using CAPI

by Nikolas Bampetas

Abstract

DNA carries all the information needed to define the genetic structure of an individual. The large amounts of information stored in the DNA makes it costly to store and to process. In this work, a fast and efficient implementation of a Field Programmable Gate Array (FPGA) based, streaming multicore architecture for accelerating variant calling algorithms will be designed. We focused on the HaplotypeCaller which is the variant calling software part of the Genome Analysis Toolkit (GATK), which is one of the most widely used DNA analysis tools in the field. The most time consuming part of the HaplotypeCaller is the PairHMM algorithm. PairHMM is a probabilistic algorithm that executes pairwise alignment of two sequences. Starting from an existing single core PairHMM accelerator design, which was implemented using the POWER8 Coherent Accelerator Processor Interface (CAPI), we designed three extra cores of the PairHMM algorithm that can work independently and increase the performance of the overall system. The new accelerator achieves a 2.2x speedup in comparison with the single core. The accelerator is integrated with the HaplotypeCaller and uses CAPI to access a shared processor-accelerator memory for direct communication. A JNI call is used to send the memory addresses to the accelerator, which reduces the communication overhead between the HaplotypeCaller and the accelerator. Results show that the application is not able to saturate the accelerator with data, resulting in accelerator idle time and under-utilization. Overall, the accelerator incurred a total idle time of 26% of the runtime for the different data sets. It would be beneficial to implement a faster version of the HaplotypeCaller which can load data sets to the accelerator as current data sets are being processed.

Laboratory : Computer Engineering
Codenumber : CE-MS-2017-12

Committee Members :

Advisor:	Zaid Al-Ars, CE, TU Delft
Chairperson:	Zaid Al-Ars, CE, TU Delft
Member:	Stephan Wong, CE, TU Delft
Member:	Rene van Leuken, CAS, TU Delft
Member:	Johan Peltenburg, CE, TU Delft

Dedicated to my family and friends

Contents

List of Figures

Acknowledgements

I would first like to thank my thesis advisor Johan Peltenburg and Shanshan Ren of the computer engineering lab at TU Delft. The door to Prof. Zaid Al-Ars office was always open whenever I ran into trouble or had a question about my research or writing. He consistently allowed this thesis to be my own work, but steered me in the right the direction whenever he thought I needed it.

Nikolas Bampetas
Delft, The Netherlands
November 14, 2017

1 Introduction

1.1 Context

DNA analysis has experienced great developments over the last century. Many researchers are focusing on new algorithms for DNA analysis, that use large scale data sets as input, making them rather expensive in terms of execution time. Scientists try to reduce the execution time of these algorithms by following different strategies and enabling hybrid solutions.

DNA analysis is a important because it can offer us information that can be used in a wide range of fields, ranging from improving plants and livestock traits to human health and disease diagnosis. In the agricultural field, for example, DNA analysis can offer us insight to improve the traits and value of plants [1]. Through DNA information that will be extracted from DNA analysis, some environmental variables can be modified to achieve different results in the growth process. This way we can model plant development in order to predict the reaction for different environmental variables.

In terms of human social value, DNA genetic research is currently used to find ancestry and to identify criminal activities. People now can identify if other individuals are blood related to them by using DNA testing methods. In terms of health, during pregnancy, doctors can request blood testing for children to identify and prevent illnesses and conditions that may face the newborn child. DNA testing can also be used to check if people are infected with viruses or to diagnose the type of cancer they may have.

Species origins can also be traced by DNA testing and many researchers and scientist use this method to discover the origins of many species and their evolution during the ages.

In the 21st century, DNA is used regularly by many fields to provide insights that can improve the outcome of some procedures. For example in forensics, DNA is critical as it can pinpoint the relation between individuals and specific criminal activity. This is possible because the DNA of an individual represents a unique fingerprint, enabling unique identification of individuals.
The examples above illustrate the importance of DNA and the value DNA analysis can add to society.

The analysis is a collection of different steps and procedures. An important step is DNA variant calling that is responsible to find out variations (also called, muta-

tions or differences) in different DNA sequences. The variant calling procedure can be performed through different algorithms which follow different approaches to achieve the same outcome. In this project, we will use a probabilistic model for sequence comparison algorithm called PairHMM. In addition, this project will develop a multicore hardware accelerator of this algorithm in order to minimize the execution time of the overall system. To be more precise, the hardware that we will use, gives the opportunity to an external hardware to request and receive data from the cache of the machine without the contribution of CPU which will reduce the latency of the application.

1.2 Problem statement

DNA contains important information for an individual. However, DNA information is quite big to store and analyze, since one gram of DNA needs 215 petabytes of storage space [2]. This large amount of data stored in the DNA increases the cost of analysis making it inaccessible for various applications. Therefore, it is important to develop fast and cost-efficient algorithms that can calculate different sequences in a smaller amount of time.

As explained above, DNA algorithms can be expensive in terms of execution time because the large amount of data stored in the human DNA. In addition, these algorithms need many iterations to complete the execution for the whole DNA and to perform all the required analysis steps.

The large amounts of data needed to store DNA information resulted in the need to develop efficient ways to reduce the execution time of those algorithms while keeping the high quality of the results.

Parallel programming has been introduced, which changes the way that people develop an application. Now the trend is to try to utilize all the hardware capabilities in order to perform parallel calculation in the same cycle. The parallel execution for big data sets can provide a reduction of the overall execution time of the program.

Field programmable gate arrays (FPGAs) can be candidates to achieve this high parallelization. FPGAs are an integrated circuit designed to be configured by a customer or a designer after manufacturing. This means that they can be reconfigured at any time to implement different solutions for different problems. The different hardware architectures can reduce the execution time of the application. FPGAs used for prototyping and for many different problems. Their advantage lies in the fact that they can be updated with a new hardware design easily.

Even higher performance can be achieved by designing of a system that combines both high performance computing and hardware acceleration. This hybrid design could offer high throughput as the software can make all the less intensive calculations in terms of performance while the accelerator performs the heavy calculations.

In our project, we will use an existing FPGA design and create a high-throughput accelerator system in order to accelerate the PairHMM and achieve higher throughput and higher power efficiency for this computationally intensive algorithm. The goal is to create multiple different independent cores of PairHMM algorithm using the Coherent Accelerator Processor Interface (CAPI).

1.3 Research question

A Hybrid approach will be followed in this thesis by combining software and hardware for our problem. Our main goal is to increase the number of cores of an existing hardware design that uses CAPI and integrate it to software. The questions that should be answered at the end of this project are:

- How efficient could such a hybrid system be?
  Could hardware and software combination offer better performance than a standalone application or there will be problems that will make the system slower.
- Is the software capable of saturating the hardware accelerator?
- Should we make the application faster?
- Could we achieve higher throughput by creating a multicore accelerator?
  By creating more cores of the algorithm can we achieve higher throughput in the system? Will the software saturate the accelerator?

1.4 Contribution

The list below discusses the contributions of this thesis project.

- Evaluation of different hybrid systems for DNA analysis applications
- Extension of the accelerator design with extra cores
- Implementation of synchronization mechanism of the design for efficient execution.
- Modification of HaplotypeCaller collect the dataset and send it to accelerator.
- Implementation of independent functionality of HaplotypeCaller and accelerator
- Use of sun.misc.unsafe to HaplotypeCaller to manage memory reads and writes.
- Use of JNI calls to call a C application that prepares the final data for the accelerator and enable it.
- Implementation of load balancing mechanism in C application to distribute effectively the data to the different cores.

By achieving these contributions, we are able to integrate HaplotypeCaller with the accelerator, offering a mechanism for the correct and efficient distribution of data to each core in the accelerator. In addition, we use two technologies sun.misc.unsafe and JNI call to achieve the fastest possible communication between the different layers of the system.

1.5 Thesis outline

The present thesis has been divided in three main parts. The first part, consisting of Chapters 1-3, is concerned with problem definition, project description and literature review of prior relevant work. In Chapter 2, description of DNA, variant calling algorithms are discussed and presented as motivation to this project. Chapter 3 presents the existing accelerator, its structure and the methodology to be followed for the integration of HaplotypeCaller with the accelerator.

The second part of this thesis, consists of Chapters 4-6 and represents the core of this work. In Chapter 4 different solutions and their characteristics and limitations are presented, to identify the most suitable technology or combination of technologies for the acceleration of HaplotypeCaller. Chapter 4 also includes a comparison and selection of the fastest solution alternatives. In addition, this chapter introduces the concept of integration with different tools that allow the conversion of the data that is managed by a virtual machine to the native environment.

In Chapter 5, simulation results and findings are presented and analyzed in the third and final part of this thesis. Based on the derived conclusions, recommendations on future work are given.

2 DNA analysis algorithms

In this chapter, we present different algorithms for DNA alignment and their functionality. We also present a collection of different approaches that have already been implemented, with target to increase the performance and maintain the accuracy of those algorithms. By the term performance, we mean the reduction of the total execution time by following different implementations of alignment algorithms and different hardware architectures. Before we proceed with the algorithms we should describe the DNA molecule and its structure.

Frederich Miescher is the first human that observed the DNA in 1869 [3]. Before 1953 other scientist could not understand the importance of DNA. In 1953 James Watson, Francis Crick, Maurice Wilkins and Rosalind Franklin understood the structure of DNA [3] as a double helix. They understood the importance of the DNA and the information that it contains for each organism and in 1962 they were awarded with the Nobel Prize in Medicine for their discovery.

2.1 The DNA molecule

DNA is a molecule, that carries all the information an organism needs to live, reproduce and develop. This information is inherited from the parents of the organism and can be found in each cell of the organism [4]. Researchers could extract much information from DNA that will help in the development of biology and could offer many insights for many diseases that humanity has been trying to face for a long period of time.

2.1.1 DNA structure

DNA is structured as nucleotides that contain a phosphate, a sugar and a nitrogen group. Adenine (A), thymine (T), guanine (G) and cytosine (C) are the four different types of nucleotides that encode the information in the DNA. The order of nucleotides in the DNA is what defines the information of the organism in the same way the order of letters in the alphabet can be used to form a word. The order of nucleotide bases in a DNA sequence forms genes, which in the language of the cell, tells cells how to make proteins.

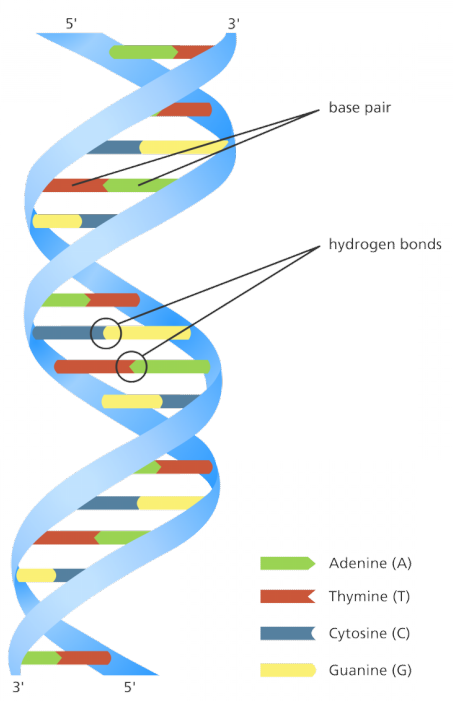


Figure 2.1: Structure of DNA (Rachael Rettner, 2013)

The double helix is the outcome of the attachment of nucleotides in Figure 2.1. Nucleotides are attached together to shape two strands. At the side of the helix, the phosphate and sugar molecules can be found while inside the helix are the bases. The bases on one side of the helix combine with specific bases on the other half of the helix as follows: adenine pairs with thymine, and guanine pairs with cytosine.

2.1.2 DNA sequencing

DNA sequencing is the procedure that a computing system figure out the order of bases in a sequence. Technology is used to identify the order of bases in genes, chromosomes, or an entire genome. Researchers in 2000 mapped the first sequence of the human genome [3]. This was a breakthrough for the field of computer science and medicine. The reason is now the combination of those two fields can help researchers to understand the human organism faster. Now real time simulations can be held in order to understand the reaction of an organism. By that way, new treatments can be developed and many lives can be saved.

DNA sequencing creates a new need in the field of computer engineering and computer science. New aspects should be created in the way that scientist investigate DNA. Researchers tried to map all the different DNA sequences for a various number of illnesses and viruses. By checking if DNA matches with the collected illnesses DNA sequences database, it can be defined what is the cause or the illness of an individual. The above

need created a new field which merge two different field biology and informatics (bio informatics).

2.2 DNA alignment

Algorithms have been developed in order to match different DNA sequences. These algorithms are named sequence alignment algorithms. Sequence alignment is the procedure to identify segments of similarity by arranging sequences. The rows of a matrix usually represents aligned sequences. DNA alignment uses the term gaps which are placed between the residues so that the same characters of two different sequences are aligned in the same columns. The most important and effective DNA alignment algorithms are Needleman-Wunsch and Smith-Waterman.

2.2.1 Needleman-Wunsch algorithm

Global alignment

Global alignment [5] is an alignment method that does an end to end alignment of target and query. Before we proceed with the algorithm of global alignment, it crucial to define some important elements that contribute to the final alignment.

Gaps are presented by multiple dashes. Gaps could be placed in both sequence with target to offer aligned the two sequences. The alignment of a letter with a null is handled by the algorithm as an insertion of a letter into one sequence, or the deletion of a letter from the other sequence. A letter aligned with a gap is called indel.

Alignment scores are the total sum of scores for aligned pairs. The alignment can be either with letters either with letter and gap.

Substitution scores are called the scores for aligned pairs of letters. The letter aligned letters can be identical or not. To simplify it substitution scores are produced by match and mismatch scores.

A pseudo code that describing the how global alignment is performed:

DEFINITION:
$s(a, b)$: is the substitution score for two aligning letters.
g: is the gap score.
X_i: is a part of a sequence produced by the first i letters of X sequence
Y_j: is a part of a sequence produced by the first j letters of Y sequence
$SIM(i, j)$: is the score that presents the similarity between X_i and Y_j.

PSEUDO CODE:
Result: Similarity(X,Y)
for *i = 0,...,m* **do**
| SIM[i,0] = i*g
end
for *j = 1,...,n* **do**
| SIM[0,j] = j*g
end
for *i = 1,...,m* **do**
| **for** *= 1,...,n* **do**
| | SIM[i,j] = max(SIM[i-1,j-1] + s(X[i],Y[j]), SIM[i-1,j]+g, SIM[i,j-1]+g)
| **end**
end

Algorithm 1: Needleman-Wunsch algorithm

In Algorithm 1 the first and second loop are used in order to initialize some values on SIM array. The last loop is responsible to check if there is a match of a mismatch between certain letters of the sequences and assign the correct score.

Semi-global alignment

The NW semi-global alignment is following the same principles as the global alignment. The only difference is that for semi-global alignment do not take into account the gap penalties which appear at the beginning or at the end of either sequence and the Algorithm 1 is modified in the last loop segment in order to discard those gaps.

2.2.2 Smith-Waterman algorithm

On the other hand, Smith-Waterman [6] algorithm is used for the local alignment of two sequences. SW is characterized as local alignment tries to maximize the alignment score and the final alignment by checking partial matches of the query and target, instead of looking at the entire sequence, in contrast with NW. The described functionality is presented in the Algorithm 2 in which the first two loops are the same as NW. The last one checks the addition previous score and the present score and if are smaller of zero it

assign the value zero.

Result: Similarity(X,Y)
for *i = 0,...,m* **do**
| SIM[i,0] = i*g
end
for *j = 1,...,n* **do**
| SIM[0,j] = j*g
end
for *i = 1,...,m* **do**
| **for** *= 1,...,n* **do**
| | SIM[i,j] = max(SIM[i,j] = max(0,SIM[i-1,j-1] + s(X[i],Y[j]), SIM[i-1,j]+g,
| | SIM[i,j-1]+g)
| | S=max(S,SIM[i,j])
| **end**
end

Algorithm 2: Smith-Waterman algorithm

2.3 Variant calling

A variant call is a result that there is a difference between nucleotide and reference at a certain position in an particular genome or transcriptome. It usually accompanied by a variant frequency and some measure of confidence. In the next sections we will describe some of them, their contribution and their functionality.

2.3.1 Pair Hidden Markov Models

There are three different application for Pair Hidden Markov Models [7]. It used for pairwise sequence alignment, for the analysis on two aligned sequences and for pairwise alignment and analysis.

In the Pairwise sequence alignment, it has an alphabet of a defined characters and we provide to the algorithm two different sequences of this alphabet. For example, let's take the alphabet that it is used for DNA which is ACGT. Two different sequences are ATGTTAT and ATCGTAC. By inserting "-" and shifting two sequences, they can be aligned in two rows with the same length and the result will be
A T - G T T A T
A T C G T - A C

A scoring mechanism should be present as we present in the previous alignment algorithms. Mismatches are scored by m , indels are penalized by s, and matches are rewarded with +1, the resulting score is:
#matches m (#mismatches) s (#indels)
The result of the previous example will be 5- m -2s.

pairHMM[8] for gap scoring using the affine gaps, so it is crucial to explain what is

the affine gaps concept. The affine gaps have an additive way to calculate consecutive gaps.

- -r-s when there is 1 indel
- -r-2s when there are 2 indels
- -r-3s when there are 3 indels, etc
- -r- xs (-gap opening - x gap extensions)

This scoring procedure is reducing penalties as it uses a weighted way to calculate the final score.

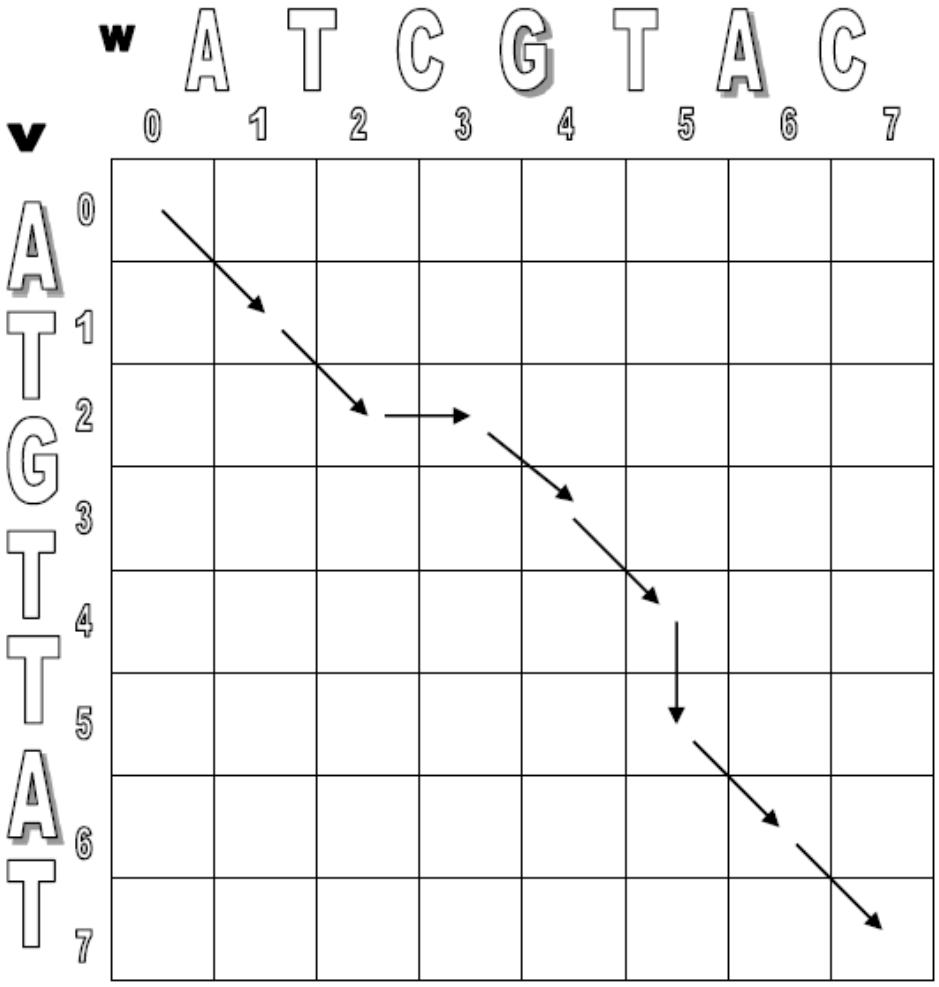
HMM for sequence alignment uses affine gap scores for hidden states of the two sequences.

- Match (M)
- Insertion in x (X)
- insertion in y (Y)

Observation states

- Match (M):$\{(a, b)|$ a,b in S $\}$
- Insertion in x (X):$\{(a, -)|$ a in S $\}$
- Insertion in y (Y):$\{(-, a)|$ a in S $\}$.

Now that we have the all the constraints of pairHMM let's see an example of an alignment of the previous example.



A T - G T T A T
A T C G T - A C
M M Y M M X M M

After the alignment, we should start the procedure to get the scoring of the the alignment. First of all, we have the
Finite State machine:
M:(+1,+1)
X:(+1,0)
Y:(0,+1)
Emission Probabilities:
M: $P_{xi,yi}$
X : q_{xi}
Y : q_{yi}

Lets elaborate to the above calculations and how can help us to conclude to our final result. Based on the HMM, a probability score is assigned to the alignment of two sequences. We call observation symbol of the HMM when there is an alignment between two letter of the two sequences or of a gap and a letter like it was described in the example. The next step is that it should represent a scoring technique reflecting an iterative model. The probability of each aligned pair is determined by the transition and emission probabilities. Finally HMM is looking for the maximum probability alignment of the two sequences that are given as inputs to the algorithm.

In transition probabilities we annotate as the probability for first gap and as probability for extending gap. In addition we should present the variables that will be used for each occasion for the emission probabilities. Match probabilities is annotated as $P_a b$ while Insertion in x with q_a. Insertions in y will presented with q_a-.

The scoring alignments follows the some rules. For pairs of x and y with length of m and m accordingly, there are multiple alignments of them which every one can be found to another state of sequence with length between maxm,n and m+n. Providing the emission and transmission probabilities, there is a fixed score for every alignment and is the product of the corresponding probabilities. If there is at least an alignment it will be the one that maximizes this score, the same principle that is followed in the most sequence alignment algorithms.

The next step of the algorithm is to find the most probable alignment. To achieve that it uses the following concept and formula. Let $v^M(i,j)$ be the probability of the most possible alignment of x(1..i) and (1..j), which ends with a match. Similarly, $v^X(i,j)$ and $v^Y(i,j)$, the probabilities of the most probable alignment of x(1..i) and y(1..j), which ends with states X or Y, respectively.

$$v^M[i,j] = P_{x_i,y_j} max = \{ (1-2)v^M(i-1,j-1), (1-)v^X(i-1,j-1), (1-)v^Y(i-1,j-1)$$

The final step is to add to our algorithm termination probabilities. The reason is that there will be alignments of the two sequences that have different lengths. To have a functional probabilistic model we should define a probability to assign on the different sequences of different lengths. To achieve that it should be added an end state in the algorithm, with transition probability t from any other state to end. This assumes expected sequence length of 1/ t. The last transition in each alignment is to the end state, with probability t.

At this point, we should notice that there are many different approaches of pairHMM algorithm. The aforementioned is more than enough in order to understand the aim of the project.

2.3.2 Genome Analysis Tool Kit

The Genome Analysis Tool Kit (GATK)[9] is the tool that used by the industry with target to identify single-nucleotide polymorphisms (SNPs) and indels in DNA and RNA

sequences. GATK is developed to take into account somatic variant calling tools and to face copy number and structural variation. GATK also contains many options to execute tasks for the processing and the quality control of a high-throughput sequencing data.

There are different callers in GATK that were designed to calculate exomes and genomes generated with Illumina sequencing technology [9]. The best part of GATK is that offers adaptability to different approaches and designs as it was developed to be flexible in changes. At the beginning was used for human genetics and now has evolved for any organism, with any level of ploidy.

Let's now describe the best practice of GATK. When DNA is being studied in a lab, can't be treated as isolated and disconnected tasks. Every task is a part of a procedure, developed in order to optimize yield, purity and to guarantee the reproducibility, consistency of all samples and experiments. The same should be happened with sequencing data as well.That's why GATK offers applications for reads-to-results workflow, which is tested in production at the Broad Institute[9] and optimized to provide the most precise and accurate results with the fastest in terms of computational time and complexity ways.

The GATK is designed to work on Linux and MacOS X. Windows and android are not supported and there isn't plant to be developed in those platforms. GATK needs Java 1.8 or newer version installed in the system, and some tools additionally require R to generate PDF plots. Versions of GATK up to 3 were optimized to run in research computing environments like clusters and servers. The next generation of GATK tool is being developed to run best in cloud environments and to Spark architectures wherever possible.

Figure 2.2: GATK overview by "https://software.broadinstitute.org/gatk/"

2.3.3 haplotypeCaller

The haplotypeCaller [9] is capable to call indels and single-nucleotide polymorphisms (SNPs) at the same time via local de-novo assembly of haplotypes in an active region. To elaborate more, when there is a region with variations, it deletes the existing mapping and starts to restructure the read in this segment with result to increase its accuracy for calling regions that are difficult to call by definition, for example when they have many types of variants close to each other. It also makes the haplotypeCaller more efficient at calling indels than position-based callers like UnifiedGenotyper.

There is in haplotypeCaller a mode called GVCF and it is used for scalable variant calling in DNA sequence data, Its operation is to create an intermediate genomic gVCF by running for all samples. The intermediate genomic is used for joint genotyping of multiple samples in a way, which enables high performance processing of samples as they come out the sequencer.

In addition, haplotypeCaller is capable to handle non-diploid organisms as well as pooled experiment data it is crucial to mention that the algorithms used to calculate variant likelihoods is not efficient to calculate extreme allele frequencies, so its use is not recommended for somatic variant discovery like cancer. Finally, haplotypeCaller can handle the splice junctions of the RNAseq procedure which is a challenge for most a lot of existing callers.

It is crucial now to describe the main parts of haplotypeCalles in order to identify the reasons why we should develop an accelerator and how it will help the overall system to become faster [9].

1. Define active regions. Based on the variations the caller identifies the active regions that should be processed.

2. Determine haplotypes in active region by assembly. After caller defines the active regions, it executes a De Bruijn-like graph to reassemble them, and investigates for possible haplotypes in the input data. The next step of application is to align again the haplotype with the reference haplotype by executing Smith-Waterman algorithm with target to identify variant sites.

3. Determine likelihoods of the haplotypes given the read data. For each ActiveRegion, the application performs a pairwise alignment all reads with all haplotypes using the pairHMM algorithm, which is heavy in calculations and decreases the overall performance of haplotypeCaller. The pairwise alignment has as a result a matrix of likelihoods of haplotypes given the read data. These likelihoods are marginalized in order to find out the likelihoods of alleles for every variant site given the read data. This is the part of the haplotypeCaller that we are going to replace with the accelerator as it is the most time consuming and the one that we are going to implement using CAPI is the third one.

2.4 Related Work

The importance of DNA drove scientist to develop methods to analyze DNA efficiently and with high performance. In the following, we present some related work in the literature concerned with developing high performance solutions for DNA analysis.

GATK provides a pipeline that can perform DNA analysis, however as it was mentioned previously, the DNA information needs large data sets of hundreds of

gigabytes. For this reason, increasing the performance of the tool is required.

Big data approaches were used to increase the performance of the GATK pipeline as the amount of data required can be a candidate for big data approach. SparkGA [10] is a parallel solution for DNA analysis that uses the Spark framework and GATK. This project exploits the possible data parallelism with an effective load balancing of data on different nodes. This solution is designed in a way that allows the utlization of nodes with a limited amount of memory size as low as 16gb per node. The experiment was implemented with 20 nodes and it achieved 75% increase in performance over the state-of-the-art application and with accuracy 99.9%. There are more project that tried to accelerate the GATK with heterogeneous solutions [11]. There is also another cluster based big data solution for GATK pipeline that achieved 63% faster execution than a hadoop based project [12].

Power efficient solutions were introduced for various DNA analysis algorithms, for example BWA-MEM algorithm [13]. BWA-MEM algorithm performs genomic data mapping. To identify a power and performance efficient solution, comparisons were performed on a FPGA implementation on Xilinx Virtex-7 device and on a GPU implementation using an NVIDIA GeForce GTX 970. The comparison had as a reference point the results of the sequential application of the BWA-MEM algorithm. The FPGA implementation proved to be 4x more power efficient in comparison with the GPU implementation which is a result of the difference in the frequencies of the two different devices. The performance of both solutions was 4.1x higher than the baseline model.

Other projects focused on accelerating the BWA-MEM algorithm. A GPU implementation was implemented on the Seed Extension phase which is a time consuming segment of the algorithm that can consume 50% of the total execution time of the application. The GPU-based model achieved the maximum theoretical speedup with the use of systolic arrays [14]. For the BWA-MEM algorithm, many other solutions attempted to reduce the power consumption and increase the performance of the algorithm like [15],[16].

In addition, heterogeneous approaches also attempted to increase the performance of the GATK pipeline. A heterogeneous system is a system that uses different types of hardware to optimize the execution time. An example of a heterogeneous system is the use of a convey HC2ex platform for the BWA-MEM algorithm that attempted to speedup the local alignment and sufx array lookup algorithms. The heterogeneous implementation contains 2 cores, one for each algorithm and the speed up realized is 2.8x for local alignment and 5.7x for lookup. The software side of this project was optimized as well, offering a speedup of 1.7x on the software side and an overall speedup of 2.6x to the overall BWA-MEM [17]. Another project using systolic arrays design for the local alignment algorithm achieved 96% of the maximum theoretical speedup. The result was a 45% speedup of the local alignment in comparison with the software

solution [18].

Another part of the DNA alignment is that read bases are becoming bigger, which results in increasing the read alignment execution time because of the read bases growth. The read alignment can be performed with two different methods with seeding or with extension. An open source suite was developed to identify the best seeding and extension combinations for read alignment [19]. The result was that for the extension local alignment is 3.6x more accuracy in comparison with other techniques. The combination of the BLAST seeding algorithm with the local alignment achieves 6x more accurate data. [19] presents a vectorized implementation of local alignment that runs 4.5x time faster in comparison with the baseline model.

Below, we describe related work for the pairHMM algorithm and the HaplotypeCaller. We will investigate implementations that increase the performance of the HaplotypeCaller while the maintaining the same accuracy.

An accelerated version of pairHMM on HaplotypeCaller was created by implementing the algorithm in GPU. GPUs could be great candidates to accelerate a system because of their architecture. They contain multiple streaming processors that initially were designed for graphics processing. An GPU based acceleration with two different strategies achieved an accelerated version of pairHMM. The two different strategies that were followed, are inter-task parallelization and intra-task parallelization [20]. The accelerator was an NVIDIA Tesla K40 which contains 2880 cores that could run at 750 MHz. For the host-side of the application, a POWER8 machine was used that contains 20 cores. The inter-task parallelization method is to map all the pairHMM algorithms to a thread so each core of the GPU can run in parallel multiple versions of the algorithm. The intra-task parallelization exploits a characteristic of the algorithm and its data dependencies.

In the inter-task parallelization approach, extra optimizations where implemented. One such optimization is the sorting optimization, in which the threads in a block (wrap) where sorted with respect to the length of the reads and haplotypes. Another optimization was the increase of the computational tile size in order to avoid the memory communication overhead. The outcome of this work was that the intra-task implementation achieved throughput of 23.56 GCUPS and it was 4.82x faster than the baseline model on the 20 cores POWER8 machine.

Another accelerated version of pairHMM introduced on a FPGA device. The implementation of the pairHMM algorithm to a FPGA was proved faster than the CPU-based by offering a 27x speedup over a 32-core CPU for the BLAST aligner. The implementation of SAMtools on FPGAs achieved an overall speedup of 2.93x over the original version of CPU based SAMtools [21]. FPGA solutions were proved good candidates for DNA analysis algorithms and new works were presented for accelerating the forward algorithm using FPGAs. In one of the solutions, systolic arrays (SAs) were used for the core calculation of the pairHMM algorithm. SAs were introduced by

H.T. Kung and C.E. Leiserson [22] and are used mostly for matrix calculations. SAs consist of multiple processing elements (PEs) where one PE pass its result to the next one. In [22], pairHMM was mapped on SA arrays. The experiment was performed on a Convey HC2ex platform which contains a four-core dual socket CPU and four Xilinx Virtex-6 LX760 FPGAs. The achieved speedup of this project was 67x by using 91 PEs.

The main drawback when SAs are used for the pairHMM is that they can suffer from under-utilization. This has a number of causes: 1. padding some sequences to make all sequences have the same length, or 2. because of the control mechanism that synchronizes the units during the execution [22]. A new approach of accelerating the above design implemented by using a new interface to accommodate the design. The proposed architecture maximized the efficiency of the SA and offered 2.5x speedup than the state-of-the-art FPGA implementation and 10x faster than a state-of-the-art CPU [23].

This last implementation was the most efficient related work investigated in this thesis. Therefore, we will develop the work in [23] in order to create a multicore accelerator for the pairHMM algorithm in which each of accelerator cores is able to run independently.

3 PairHMM accelerators

This project aims at enhancing the performance of PairHMM algorithm. A hardware accelerator is provided by computer engineering lab of TU Delft[23] which was designed alongside with Coherent Accelerator Processor Interface (CAPI) interface of IBM. In the first section, we will discuss what is CAPI, how it works and what are the advantages in comparison with a mainstream accelerator. The second step is to describe the structure of the baseline accelerator and its functionality. Finally we present our different solutions and their advantages and disadvantages in terms of performance. The problems that was faced and the selected solution are discussed also in this chapter.

3.1 The CAPI interface

In this section, our target is to explain the structure and functionality of CAPI and its advantages [24].

3.1.1 CAPI description

The basic reason that drove IBM to investigate new solutions to increase performance was the physical limits of the components of the CPU. From the early history of computer systems the main focus of the companies was to increase the Frequency of the CPUs in system. Increasing the frequency means that in the same amount of time the CPU can execute more instructions. Another fact that assisted the growth of the performance was the continuous reduction of transistor size. With smaller size we can accommodate more transistors on a CPU with result more functional units in the same area.

The physical limits that reduce the rate of the performance increase were the high temperature that was produced from the transistors. The high temperature is a result of the fluctuation of the current that passes through a transistor in a tiny area. The industry and different science fields needed high performances and the computer industry had to solve this problem. The solution was to stop change the architecture of the CPU and to combine multiple cores in one chip. This approach offers the capability of multiple execution in parallel which provides faster execution time. This drove to a new era in computer industry to the multicore era in which there are multiple cores packed in a chip with goal each core to perform a task while the other cores will execute something else. The multicore era makes the hardware design companies to try accommodate more execution units in a CPU and stop trying improve its frequency.

Another strategy that was followed in order to increase the overall performance of a system by using external hardware to assist the execution of a software. Computer engineers started to design software that could use the compute capabilities of GPUs which have a different architecture that CPU does. This architecture is suitable for performing parallel executions on a program. For more heavy-calculation systems engineers designed hardware accelerator on FPGA that offered the most of the time high performance solutions. In both approaches there was a problem that decreased the performance, the memory communication between the different devices, the CPU and the memory.

To diminish the overhead of communication IBM in 2014 presented an new system called CAPI and can reduce significantly the communication time because of a new mechanism that was integrated to it. CAPI is a framework that can accommodate a custom hardware design in a FPGA and by the new mechanism developed by IBM the custom accelerator could request and store data directly from the cache of POWER8 machine without the use of CPU, giving the benefit to the accelerator to operate as a part of an application. This functionality is efficient because the accelerator can perform calculations without interfering with execution of the CPU, so both devices can work in parallel. IBMs solution allows better overall performance with a small programming funding, allowing hybrid computing to achieve success across a much broader variety of applications.

The described advantages create a new powerful tool for various fields. CAPI customers are from many fields where the need of heavy calculation in a limit period of time is critical. Workloads with computation-heavy algorithms are a prime goal for CAPI, as it is a tool that could offer solution in challenging problems in terms of data size, the demand of high performance and heavy calculation all at the same time. An example of problem with these constraints is DNA alignment algorithms.

The marketplace for accelerating complex simulations and algorithms the usage of hardware accelerators is enormous. Potential markets for CAPI could be some of the industries and markets that use complex algorithms bio informatics is one of them markets. Bio informatics is a best candidate for CAPI because applications like brain simulation or DNA analysis need heavy calculations for enormous size of data and it is critical to achieve a low execution time.

3.1.2 CAPI structure

In this part, CAPI structure is presented in order to understand what modules are responsible for the advantageous features that it offers. The innovation of CAPI is that can be loaded to a FPGA which is ported to a POWER8 machine of IBM. In POWER8 machine there is a module called the Coherent Accelerator Processor Proxy (CAPP) which is collaborating with a unit in the accelerator called the Power Service Layer (PSL) which is part of the FPGA design and these two module can share the data in cache of the POWER8. The aforementioned collaboration makes FPGA an accelerator

function unit (AFU) in which can be loaded a custom design for a particular problem. [25]

The shared cache function that is presented by CAPI offers high performance solution because it diminish the communication time between the CPU and FPGA. In tradition system when a FPGA requests data this request is handled by the CPU which performs the address translation and send the data to the FPGA. While CPU is retrieving data from the cache uses resources that could be used for other instruction.

In CAPI framework, exists the AFU which is the part of the design that contains the algorithm who is going to be accelerated. AFU includes many other segments in order to offer high performance solutions like PSL, control unit, a DMA memory and the custom unit (CU). PSL is responsible for the communication with the POWER8 machine, while the control unit act like an arbitrator for the synchronized operation of DMA and CU. CAPI offers the capability to build your own accelerator in the AFU which can directly request data from the cache of the power8 without the need of collaboration with the cpu. The fact that cpu shouldn't take action in the communication time with the FPGA reduces the execution time and the application can work in parallel without interfering with the accelerator.

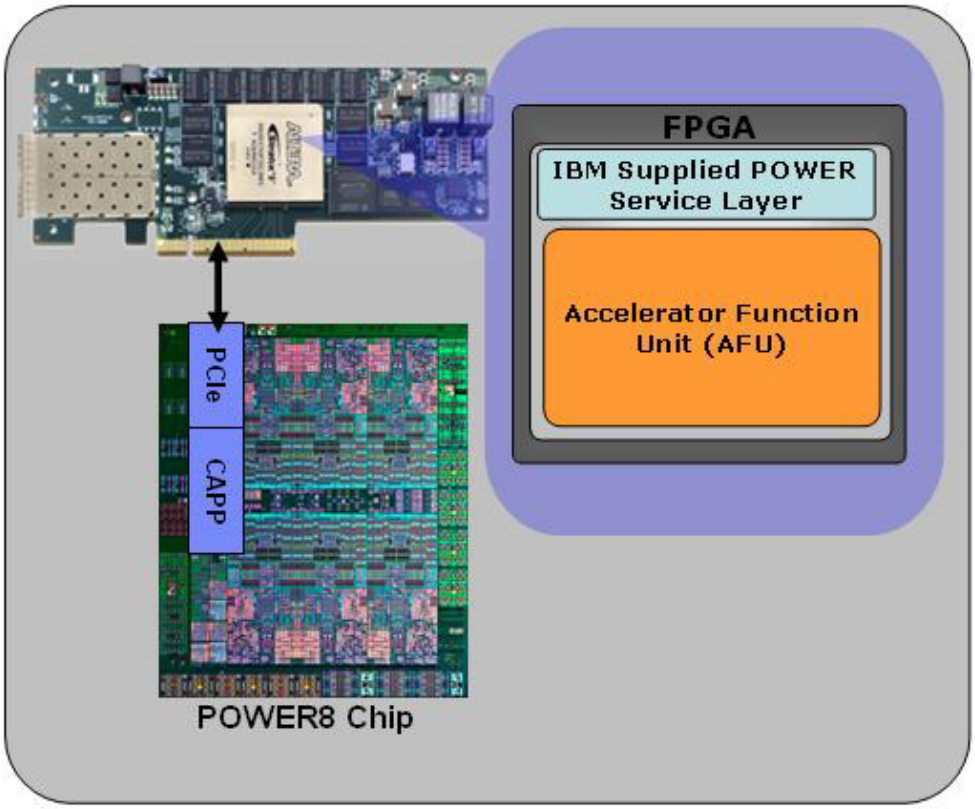


Figure 3.1: CAPI Hardware structure (B. Wile, 2014)

The FPGA is connect to the POWER8 machine through a PCI express v3 port using 16 PCIe lanes. Host can have multiple PCIe slots that can accommodate more than one AFU but there is only one CAPP in a POWER8 machine [26]. The maximum bandwidth of CAPI is limited by the bandwidth that PCIe version 3 x16 technology. The peak of the bandwidth is at 16 GB/s [27].

3.2 PairHMM on CAPI

As we present in previous sections, CAPI on POWER8 systems can provide high-performance solution for computation-heavy algorithms implemented on an FPGA. CAPI gets rid of the overhead and complexity of the input output system which makes the FPGA an extension of the CPU. The IBM solution offers higher overall performance because it gives the capability to create a custom design in it, allowing hybrid computing to be successful for many different problems for various fields. The biggest advantage that CAPI offers in comparison with a regular accelerator can be identified in Figure 3.2.

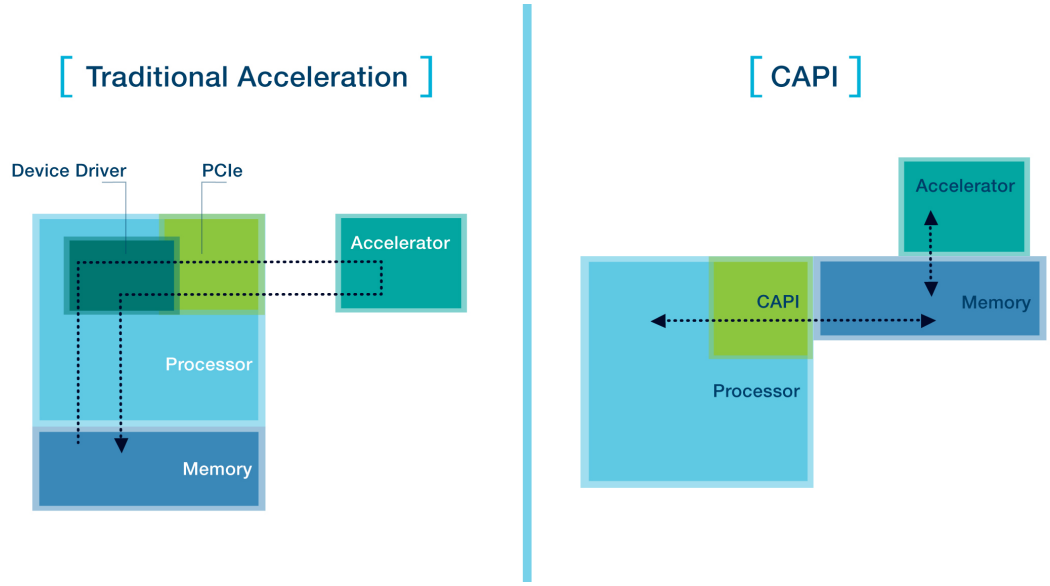


Figure 3.2: CAPI accelerator vs Regular accelerator by "B. Wile, Coherent accelerator processor interface (CAPI) for power8 system"

3.2.1 Accelerator details

CAPI was selected to implement the PairHMM algorithm because of its advantages. Now it is crucial to describe the structure of the baseline accelerator and it's functionality [23]. A diagram of the design is described in 3.3. The FPGA board that is used to be the AFU is an AlphaData ADM-PCIE-7V3. As host was used an IBM Power System S824L which has a POWER8 CPU. CAPI in order to accomodate the design FIFO where used to temporary store data that will be used as input to the PairHMM

core. By tha way it was achieved to preload a new bunch of data while an old one is processing by the PairHMM core. PairHMM core is using systolic array (SA) in order to increase more the throughput of the core [28].

The SA contains multiple pipelined processing elements (PPEs) [23]. Each PPE is responsible to execute a cycle of calculation. The architecture of PairHMM core can be compared with a 16 stage pipeline. The upper bound of PPEs that can we use in the FPGA is 112 due to DSP resources. The FPGA resources are 3600 DSP units to be utilized, but because IBM offers the PSL as placed and routed (in a checkpoint) the 25% of the DSP resources are used by PSL. The baseline accelerator was implemented by utilizing 16 and 32 PPEs with plan to develop a multicore architecture with multiple PairHMM in it that could work independently[23]. After an investigation we conclude that the maximum of the PairHMM cores are four. This is cause by the resources of the device and more specific the DSP. The baseline accelerator uses for 32 PPEs 709 DSP units and approximately 700 units are allocated by the PSL and reduces the available DSP units to 2900. The Four kernels design requires 4*709 = 2836 units which is almost to the limit of the device.

The Batch Loader is the module that is responsible to communicate with the system and to store the information of a dataset into the FIFOs, in a consecutive way while the Batch Scheduler with handshaking signals synchronizes with the Batch Loader.

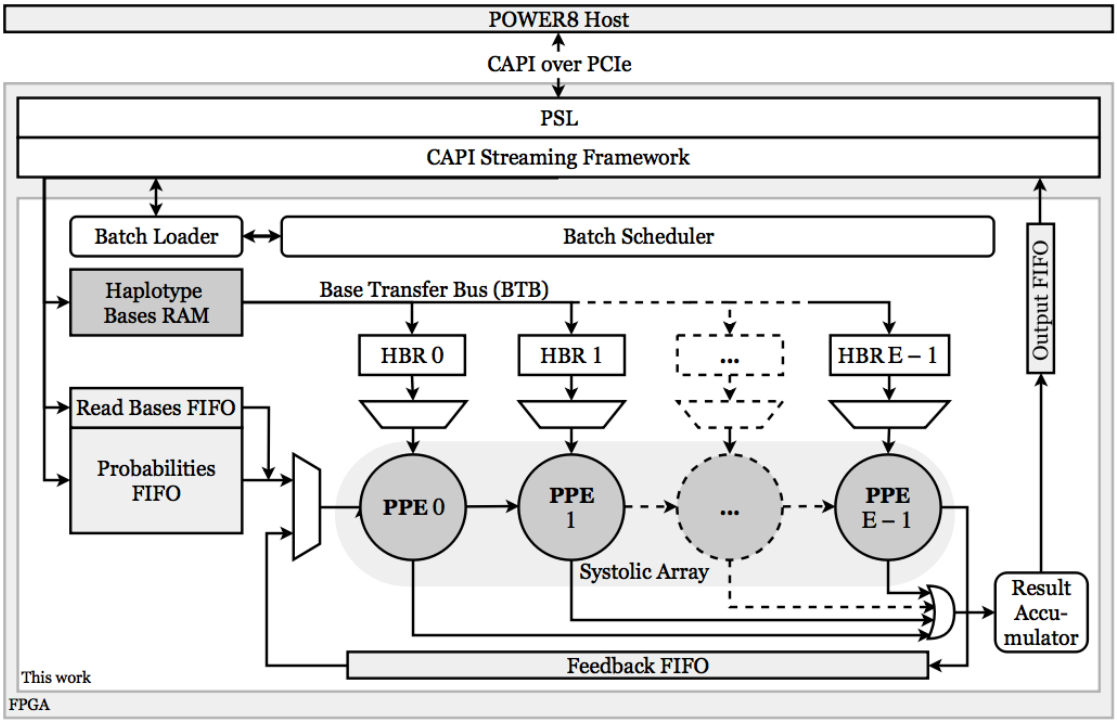


Figure 3.3: PairHMM accelerator via CAPI (J. Peltenburg et al. 2015)

Our primary goal in this project is to expand the baseline design to increase the overall performance of the system. Before we start developing the system, we should analyze and describe exactly each block how it is performing. The actual architectural hierarchy begins with the PSL signals that are connected to CAPI that is responsible for the synchronization of the CPU and the CAPI by acknowledgment signals. The communication between them depends by the state of the accelerator. For example, CAPI requests data through PSL to CAPP and PSL responds after some cycles by sending an acknowledgment and the requested data.

3.2.2 Accelerator integration

CAPI top level module is the Frame unit. The frame is responsible for the interconnection of all the functional units of the CAPI such as DMA, MMIO, Control unit. An overview can be seen in Figure 3.3 It is crucial, as it is responsible for the synchronization between the host and the accelerator interface. All the modules that are responding to host should operate at 250Mhz for terms of synchronization between both ends.

DMA or Direct Memory Access unit is responsible to handle and stores the data. Its actual function is to store data after receiving them from the host and to send data that we want to write back at host side. DMA can offer higher performance as it provides a parameter for reading and writing engines. By increasing the read and write engines we can achieve parallel reading and writing procedures at the same cycle. This option is a key modification because the goal of this project is the creation of multiple kernels accelerator of PairHMM algorithm that they run in parallel in order to fully utilize the FPGA. DMA unit uses different ram blocks for storing data something that could be a drawback on the synthesis step as it consumes area and in this design different kinds of storing blocks are utilized. In addition, it includes a smart mechanism that handles different engines for efficient read and write and it also is responsible to inform the other functional units that interact with it about the stream that is responding. Finally, DMA uses only one bus for all the generated engines. This fact could be a drawback for our design because it is impossible to request data from multiple engines the same cycle. However, the advantage of the one bus in the design is that the fanout will remain at the same level regardless of the number of generated engines.

The Control unit module is coordinating between the CAPI and our accelerator unit or CU. The functionality of this module is simple but really important. It remains idle until the starting signal arrives from the host. Together with the starting signal arrives an address that points to the working element descriptor(WED) which is a structure that contains the addresses for all the required data. Control unit starts by requesting the WED from the host. When WED arrives control goes to go state in which it fetches the data and the start signal to the CU. To achieve the project goal the control unit should be modified in order to be capable to load the four different WED

and send them simultaneously to the CU. The control unit waits until it receives a done signal from the CU that means that the accelerator has wrote back the data to the host side. This logic should be restructured in a way that could manage four different done signals for each kernel. Through the control unit DMA buses are forwarded to the CU unit.

The custom unit or CU is accommodating the PairHMM algorithm that is described in Figure 3.3. CU should be modified as well. It should be extended with three extra kernels that will have exactly the same functionality with the first one. Our target is to achieve the parallel run of those four kernels and to offer at the same time four different results. The most challenging part of this project should be to synchronize the four kernel between them and with the DMA unit.

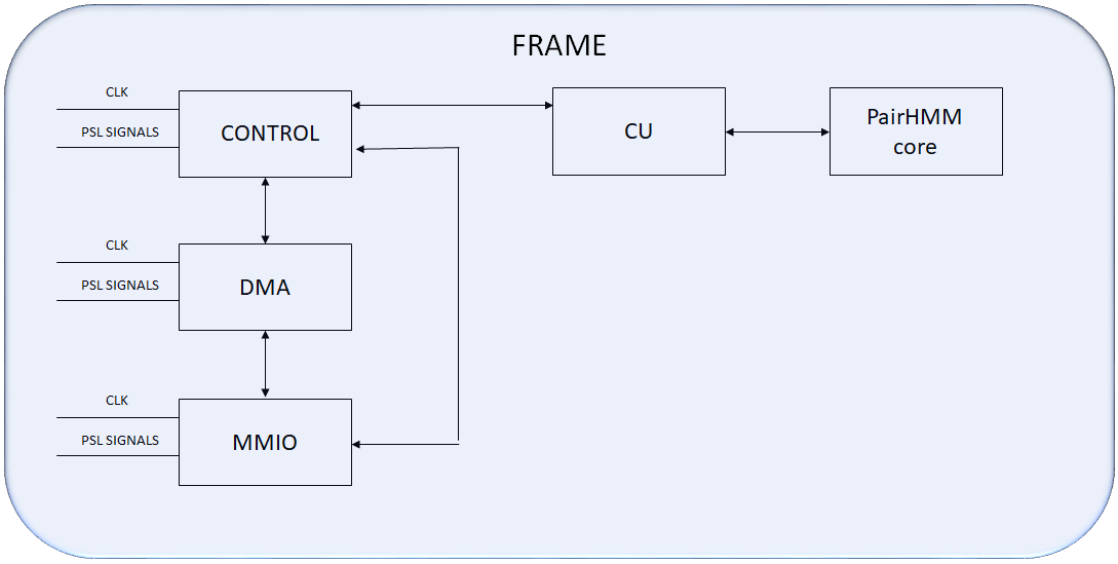


Figure 3.4: Accelerator overview

Acceleration of the HaplotypeCaller

4

In this chapter will be described the procedure that was followed in order to integrate the two project and one. The integration aims to speedup the overall procedure of the GATK-3 application for genomics. This makes the integration a challenging task as long as we should find the most efficient way to do it, with target to eliminate all the time consuming functions that are needed for this integration.

4.1 Accelerator architecture

In this section we are going to focus on the modification that should be done on the accelerator in order to increase its throughput and finally the overall performance of the system.

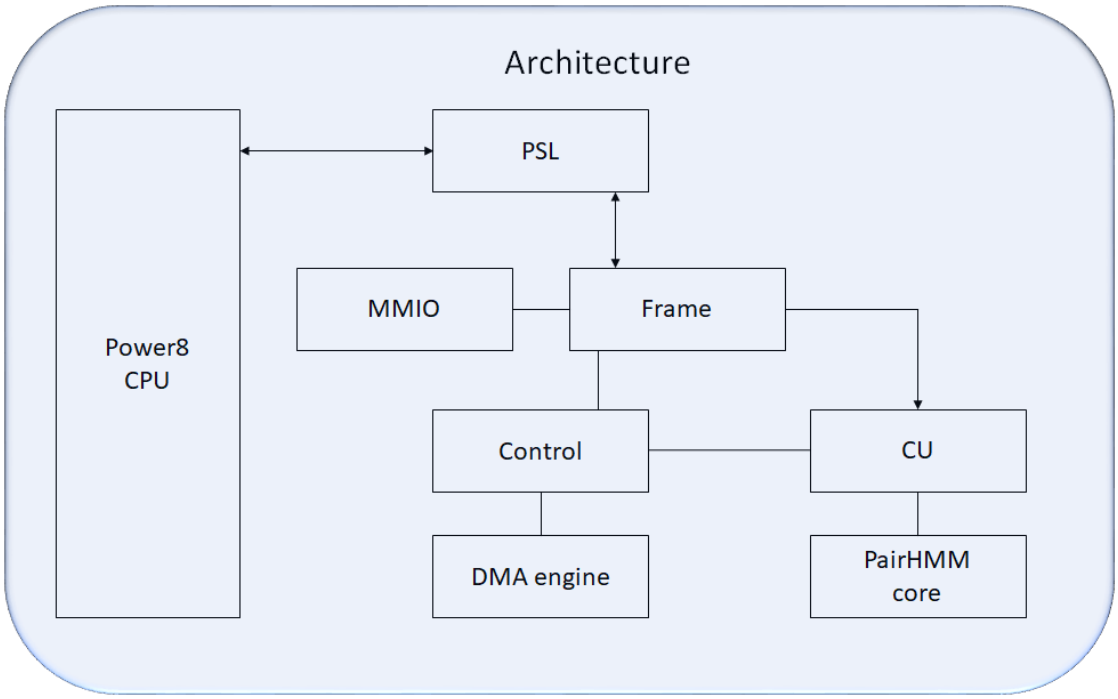


Figure 4.1: Accelerator architecture

The architecture of this design is represented in figure 4.1. POWER8 is the host side which will accommodate and serve the accelerator with the collaboration of PSL. PSL will forward all the request from the FRAME and MMIO to the host and will forward the requested data to FRAME. Frame will store the data to the DMA through

Control unit. Finally the CU will get the required data from DMA by control unit and will start the PairHMM core. Finally when the results are ready will be sent back to host through MMIO and PSL.

The target of this accelerator is to obtain communication with the cache of the POWER8 machine through CAPI. PSL is an intellectual property (IP) that provided by IBM and it is responsible for the synchronization between the FPGA and the POWER8 machine. The next layer is the Frame of CAPI. On Frame are connected all the main modules of the design. MMIO is responsible for the inputs and outputs of the peripherals. Control is handling the start of the accelerator and it is providing the the address to the CU with all the initial addresses that are needed from the Custom Unit (CU). CU is responsible for the requests from the DMA in order to collect all the data to FIFO and send them to the PairHMM core to execute the calculations. Finally, when the results are ready are written back to the cache and the accelerators waits for the next set of data. PairHMM core contains as we described PEs. The number of the PEs will define home many pairs can calculated in one run of the accelerator.

4.1.1 Accelerator solution 1

The first solution that will be implemented is the creation of three more PairHMM cores that will work independently. To achieve this functionality we would need extra DMA engines, one for each kernel. This method will allow us to calculate four different data sets of pairs simultaneously. The outline of the new architecture can be seen in figure 4.2.

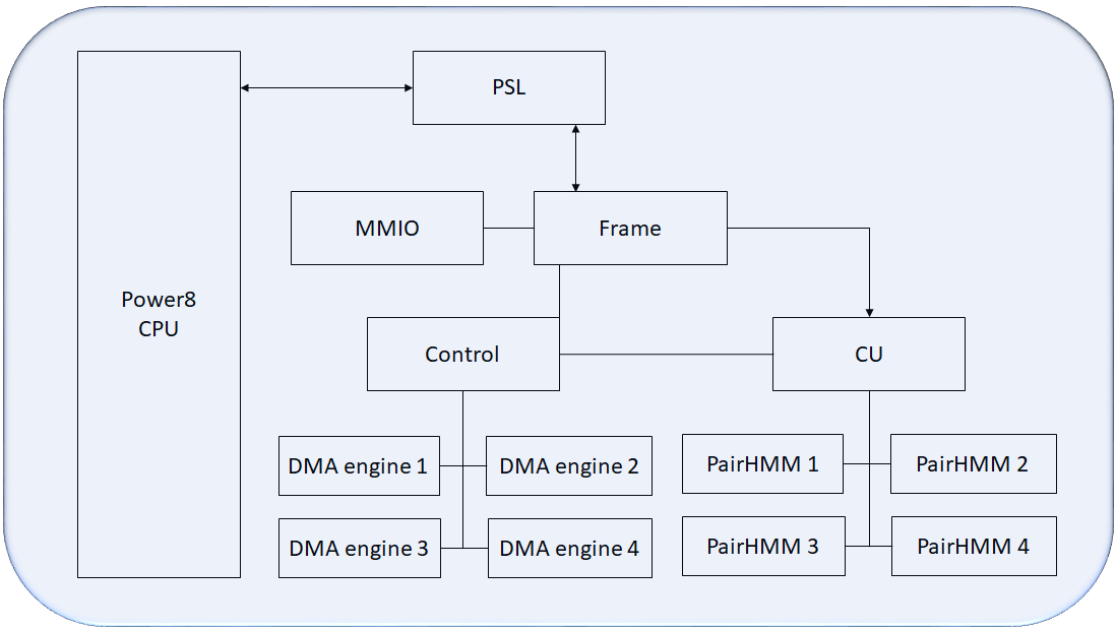


Figure 4.2: Accelerator architecture

The strategy, that was followed to increase the performance of the accelerator, is to

utilize as many resources as possible in the FPGA. However the resources are limited and the maximum of the kernels that we can create are four. DSP will be nearly 100% utilized with four kernels, so it would be impossible with the offered device to generate more than four kernels.

It is also crucial to modify multiple modules of the device in order to accommodate the three extra cores.

Control Unit

To achieve parallel execution of multiple kernels, the first step is to request four different WED that each one will be sent to an individual kernel. The data request will be handled by the control unit of the design. To implement this functionality the FSM in the control unit should be extended with three extra states and each state will be responsible to load a WED. Before we try to implement the extension of the FSM, we should figure out a technique to send the different WED together in the accelerator. The most effective way and the most scalable is by creating a list structure of WED and modified the way that hardware receives. In Figure 4.5 there are two different FSM structures. The right state machine describes the default FSM that already exists in the control unit and the left one is the modified FSM in our design.

In this paragraph, the functionality of the default FSM will be described. The control unit is in idle state and waits from the host the start signal and the address of the WED. When it receives both of them FSM jumps on wed state in which requests the WED structure with the different address that are needed by the CU and waits until it will get a valid signal from DMA that the data arrived and stored. The go state is next which gives a valid start signal at CU and waits to receive a done signal from the it. The done signal means that the accelerator is done and the results of its calculations are sent back to host. The last step is the done state which handles the resets on CAPI and final step is to go to the idle state and wait for a new start signal from the host in order to start a new cycle of calculations.

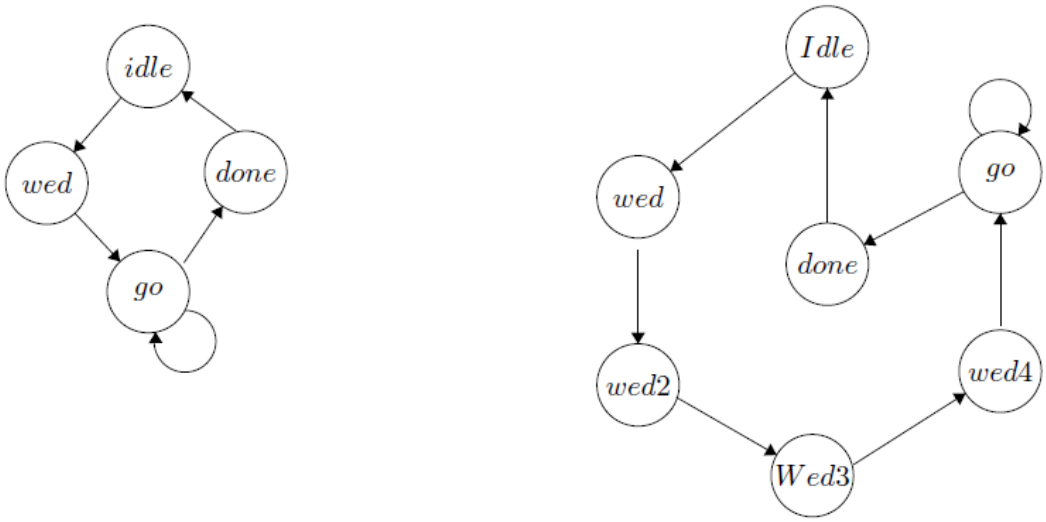


Figure 4.3: FSM of control unit

We extended the default FSM with target to achieve parallel load of multiple WEDs. The approach is similar to the default but we added some extra states. The procedure starts at idle state and waits for the starting signal and the address of the first WED from the host. The next state is wed in which we request the WED structure and when we receive them we proceed to the wed2 state to request the second WED. Exactly the same procedure is followed at states wed3 and wed4. When we have collected all WED structures and we have stored them on DMA unit we are ready to start the CU and wait until we receive done signals from CU. However, because we have multiple kernels we should wait for four different signals that each one represents a kernel. The reason why, four different signals are used, is because when a result is ready the host side should be acknowledged for performance purposes. The last state of the FSM is unmodified.

CU

In this section will be presented all the modifications that are performed but first we should give an overview of the functionality and structure of CU. The CU is the unit that accommodates the actual calculation and this will be the most crucial unit to modify. The CU waits for the start signal from the control unit. When the CU receives the start signal it stores the WEDs in registers and it starts the batch loader phase. This phase requests data by using the memory address that is included in the WEDs through different cycles. When the data are collected and prepared are sent to the PairHMM core to perform the calculations. The PairHMM core with the different FIFO that are used to store data before they are fed to PairHMM core are co-ordinated by a scheduler. The scheduler is responsible for the synchronization of the core.

The first modification was to create three extra ports of 1024bits each between the CU and control unit in order to send four different WED simultaneous. Each

WED will allocate different dataset with scope to achieve four independent execution of PairHMM in parallel. The first solution was to use the same state machine of batch loader by adding extra functionality in order to simultaneous load the a dataset to each kernel. However the structure that CU communicates with the DMA is a single bus that cannot send a request to all the DMA engines in the same cycle as we described in the previous section. This forced us to restructure our approach and to create four copies of the FSM of the batch loader that each one will handles a kernel. In addition, all four FSM should communicate effectively for synchronization purposes and because we want to ensure that only one FSM at a cycle will use the DMA. Now the first FSM loads the data for the first kernel and sends a start signal on the first PairHMM core while at the same cycle sends start signal to the second FSM to load data for the second kernel. An example of this functionality is described in Figure 4.4

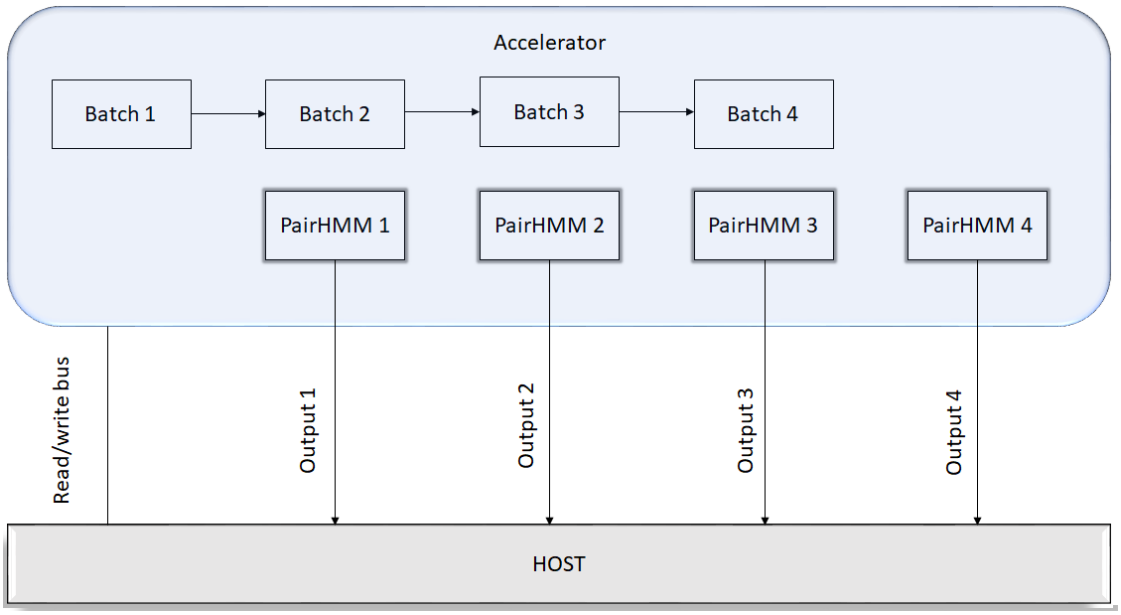


Figure 4.4: CU Batch loaders

The modified batch loader is presented in Figure4.4. The functionality of the modified design is, the first batch loader requests the first dataset that is going to be calculated in PairHMM core 1. When the data are received then the batch loader sends two acknowledgment signals. The first signal is received by PairHMM core in order to let it start processing the data. The second signal is sent to the next batch loader in order to start request and receive the second dataset for the second PairHMM core. The procedure is continuing for all the batch loaders. This behaviour is required as the CU is connected with the DMA module through one bus which cannot serve all the batch loaders simultaneously. Each batch loader need only 2% of the overall execution time and the execution time is not increased dramatically.

In addition except the batch loader modification, new Ram blocks were created to accommodate the data from each batch loader and kernel. New Haplotype RAMs, Read RAMs, probability fifos, output fifos and feedback fifos for each kernel are implemented for the independent execution. This technique will increase the area of the design but the target of this project is to fully utilize the device for higher performance so it is acceptable.

The same reason drove us to to create four distinct schedulers to handle the data in and out to each PairHMM kernel. Each scheduler works independently and is in charge of the smooth operation of the kernel is assigned on the particular scheduler. The outcome is a fully functional CAPI accelerator with four kernels. The overview of the new design is described in Figure 4.5

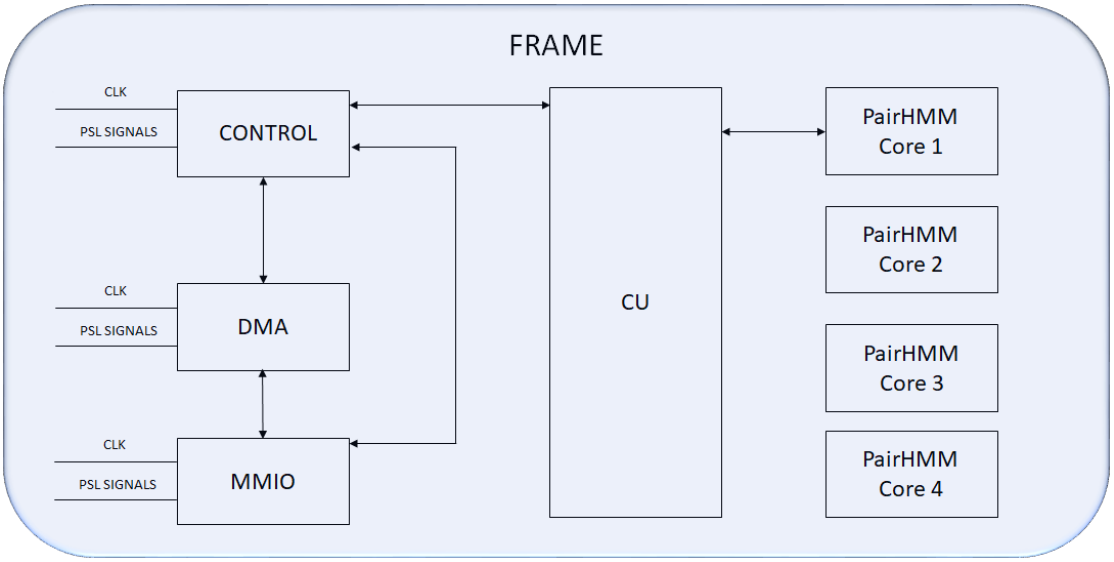


Figure 4.5: Modified Accelerator

4.1.2 Accelerator solution 2

The design that was described is functional in simulation level and the next step is to synthesize it on the FPGA board. To implement our design some procedures should be followed and their outcome will be our implementation. The first procedure is called translate and it translates our design which means that combine the given netlists and constraints into a Xilinx's design file. The second procedure is map which is responsible to fit the given design into the target device, and optionally, places the design. Place and route procedure tries to places and routes effectively the design according to the timing constraints in order to meet all the constraints and mostly the timing. The aforementioned steps are consisted by other sub procedures that will be described later. The final step is to generate Programming File. If all the previous procedures were

finished successfully then we can create the file that will reconfigure the board with our extended accelerator.

In the synthesis part, a designer can meet problems that are hard to solve. Our design has to meet some constraints during synthesis procedure physical and timing ones. The physical constraints point some blocks of the design where to be placed precisely on the device. The timing constraints are important in our design not only for performance reasons but for synchronization purposes because if some signals are not received in the specific cycle then our design will not meet synchronization constraints and will performed wrong calculations. The most important one is that host sends some signals to CAPI and wait for an answer a specific number of cycles, so both ends should operate at the same clock frequency otherwise there will be a mismatch. However, we make use of a different clock frequency in CU module something that is compulsory to meet the timing constraints as the PairHMM has multiple blocks that should run on lower frequency like IPs that performs floating point calculations. In order to meet the requirements for the write procedure we are using a slower clock of 166 MHz while the read procedure is using 250 MHz main clock.

In the modified design the total area was increased from 15% to 70%. As the design becomes bigger the algorithms that handle the mapping place and routing have a much more difficult job to do as they are heuristics algorithms and the bigger the netlist is heavier calculations are needed to find the optimal path. Congestion and timing violations were presented in the design. In this project we experienced many problems like the aforementioned but the biggest was the timing violations.

The first time that we tried to synthesize our design we experiences many timing violations with huge slacks of -1.500ns for 20.000 setups. This drove us to analyze our to design and make modification by retaining the same functionality.

The first problem was multiple timing violations of the PairHMM cores. This was an effect of multiple clock generators. We kept only one clock generator in the CU that fed all the units and blocks. All the violations that were occurred by the different clock generators were fixed by that modification. The removal of extra clock generators is beneficial in many ways for our design because it does not only fix the particular violations but at the same time, it reduces the area of the design.

The second timing violations were experienced in the CU. The CU area was massively increased and in order to manage the different modules in it, we add multiplexers in many segments of the design. The multiplexers causes a minor delay which is critical in some parts. An example of that is the switch mechanism that selects in which output fifo the results data should be written or the data that are received from the host. The only way to fix that is to redistribute the workload in more cycles and by adding registers between those states to maintain the data. The batch load FSM experienced problems because the other modules that were sending acknowledgment signals presented negative slacks. Another violation was in the state loadx of each FSM of the batch

loader where there were three addition operations. This amount of operations in one cycle could not meet the timing constraints and that force us to create a second state of loadx that we called it loadx_2 and distribute the operations in two consecutive cycles.

After all the changes that we did in the CU new timing violations where met at the DMA unit. The basic problem was on the RAM blocks when there was a signal for writing the data. The routing was so long that the period of clock was not enough to setup the the memories on the RAM blocks. To diminish that we modified the write mechanism of DMA by adding registers to store everything and write after one cycle. This reduced the routing distance and it is now possible to setup the RAM blocks in the timing period. Finally, the mechanisms of the DMA that are handling the data in DMA unit were modified in order to maintain the same functionality.

Finally, after many modifications and analysis the timing violation was transferred on the IBM checkpoint. PSL is provided in a prerouted and placed IP which is called checkpoint. PSL design cannot be modified by the user because it is intellectual property and the source code is not published. To solve the timing violation on the IBM part we followed different strategies, the first approach was to reduce the fanout of the CU. Control unit sends four data sets where each one is 1024bits. So the fanout is increased a lot only by the new three ports and will use many routing resources. This decrease the number of the routing resources for the IBM checkpoint which may causes the timing violation. After analyzing the design we checked that all the WEDs are not required at the same time so we can have a WED bus and when the CU requests a WED to send it through the bus like in figure 4.6. This reduced the timing violation even more but still the timing constraints were not met.

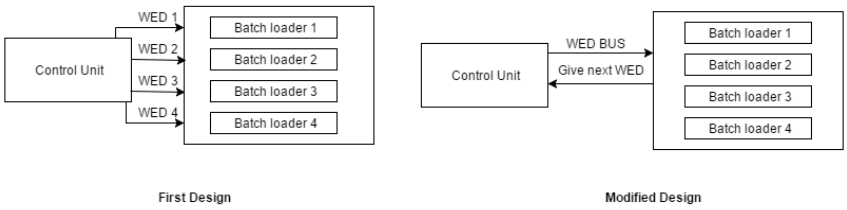


Figure 4.6: WED bus

Another change to the architecture that had a massive difference in the performance and area was the use of only one DMA read and write engine. This reduces the area a lot but created the need for a new mechanism that will coordinate the different write requests of the different FSMs in the batch loader. The requests for write procedure are performed after a signal from the previous FSM is being received. That signal will acknowledge that the writes are finished and the next batch can request for cache lines in order to write the results. The modifications just reduced the problem to IBM segment again with lower slack.

The final approach that was followed in this project was different synthesis strategies that were described in Xilinx documentation. Each stage of the flow that was described in previous paragraphs have different directives. Each directive emphasizes in a particular field like area reduction, sequential area reduction, high performance etc. To achieve the specific functionality each directive triggers different group of algorithms which are focusing on different solutions. All the previous synthesis with the timing violations were run with the directive explore. "Explore" uses algorithms that try to balance all the possible options. It is highly recommended from Xilinx however, when the timing constraints are not met then different strategies have to be followed.

The strategies are focused on five different purposes. The "performance" strategy [29] that is followed when we want to improve design performance. The area directive focuses to reduce the total number of LUT count while power is investigating the most effective way for full power optimization and reducing the power consumption of the device. Congestion strategy is recommended to be used when we want to reduce congestion and related problems.

4.1.3 Synthesis solutions

Now we will present the different directives that we used in order to diminish the problem of timing violation. We will refer the different directives that we used in each step of the synthesis procedure and we will describe each directive what outcome tries to accomplish.

- opt_design -directives:
  - Explore
  - ExploreArea
  - RuntimeOptimized
- place_design -directives:
  - Explore
  - ExtraPostPlacementOpt
  - ExtraNetDelay_high
  - SpreadLogic_high
- phys_opt_design -directives:
  - Explore
  - AlternateReplication
  - AggressiveFanoutOpt
  - AlternateDelayModeling

- route_design -directives:
  - Explore
  - NoTimingRelaxation
  - HigherDelayCost
  - MoreGlobalIterations

The Explore directive is in all cases the most demanding in calculations terms, as it combines many different algorithms to achieve the best possible solution for the design in all fields. The ExploreArea is focused mostly on the reduction of the area of the design something that could be help our design because if the area is decreased then problems like congestion are eliminated and the routing procedure could achieve better results. The RuntimeOptimized uses algorithms that trying to find the best solution for the efficient synthesize of the design. These algorithms are optimizing the parameters to achieve that on the fly.

The Vivado Design placer places gets as an input the netlist of the custom design and placing in the target FPGA in respect with its resources. Placer has multiple directives that allows the user to select different set of algorithms. Each directive optimize different target during placement. The Explore offers the best overall placement in all aspects while the ExtraPostPlacementOpt directive trying to optimize the placement after the placement is done. It spends extra time in postplacement with target to optimize even more the final placement. ExtraNetDelay_high directive increases a weight factor on the delay of each net. The increase of a weight factor on heuristic algorithms makes the procedure more demanding and the final result more accurate. Finally, SpreadLogic is optimal in cases that there is congestion in the placement of the design. The algorithms set used by Spreadlogic distributes the logic to all the available resources and they do not place them all in small amount of area.

The physical optimization stages are the most important for our project, as it is focused to solve problems like the one that we are experienced. To elaborate more, physical optimization tries to focus and eliminate the negative-slack paths of the input design from the netlist. Physical optimization is responsible for the optimization of logic of the design by making changes in the netlist and places cells accordingly. Physical optimization has two different functionalities the post-place and the post-route. The post-place mode optimizes the design based on timing estimations that are produced after the placement of the design. In post-route mode optimization occurs based on actual routing delays. In addition it will update automatically routing accordingly with the results of post route procedure. Overall physical optimization is more efficient in post-place mode, because there is the capability for logic optimization. On the other hand post-route mode is conservative in comparison with post-place mode and cannot avoid disrupting timing-closed routing. Before it starts to perform the optimization it analyze the outcome of place and route procedure and decides which of two modes will be more efficient for the current design. If there is not negative slack on the design and a physical optimization with a timing based optimization option is requested, it

exits without further optimization as the design is functional. Different directives can empower the above procedure in different manner for example AlternateReplication uses different algorithms set that replicates all critical cells. AggressiveFanoutOpt uses algorithms that are focused to reduce the total fanout of the design a result which could reduce the negative slack. Finally AlternateDelayModeling is responsible for the reduction of the net delays in the design.

The final step and the most time consuming is the routing procedure because of the heavy calculation that should perform. The router performs routing on the placed design, with target to diminish the timing violations. The router receives as input the placed design and it uses heuristics algorithms that are trying to find efficient way to route all nets in the design. The input of the router could be partially routed or unrouted or fully routed. For partially routed designs, the router uses as a reference point the existing routes. For a fully-routed design, the router evaluates the routing report and if there is a timing violation it uses a set of algorithms that tries to reroute differently this segment of the design with goal to meet timing constraints. The directives that could be used are, NoTimingRelaxation does not take into account the timing relaxation of the nets. If the router has difficulty meeting timing, it will run longer in order to achieve the timing constraints. HigherDelayCost increases the cost weight functions to increase the delay of different paths over iterations. It will need higher execution time and will offer a better routing in terms of performance. MoreGlobalIterations directive increase the global iterations to optimize the routing even if the result is slightly better than the previous iteration.

We synthesize our design with all the possible combinations of the above directives. Unfortunately, the minor timing violation was result of Explore directive in all stages. The basic problem on that occurred to the fact that the problem is not in our design but it mitigated to the Checkpoint of IBM which is responsible for the communication via PCI express port.

To conclude, we investigate different architecture solutions for the accelerator. The first one was with different DMA engines that each of them would assigned to a kernel. This approach was functional but not efficient as the DMA can service only one engine per cycle and as a result we will have a timing overhead. We could achieve the same functionality by using only one DMA engine and reducing significantly the area of the design. Finally, we met problems to the synthesis step and we used different procedures and we couldn't have a synthesized design. The most efficient design after investigation is presented in Figure 4.7.

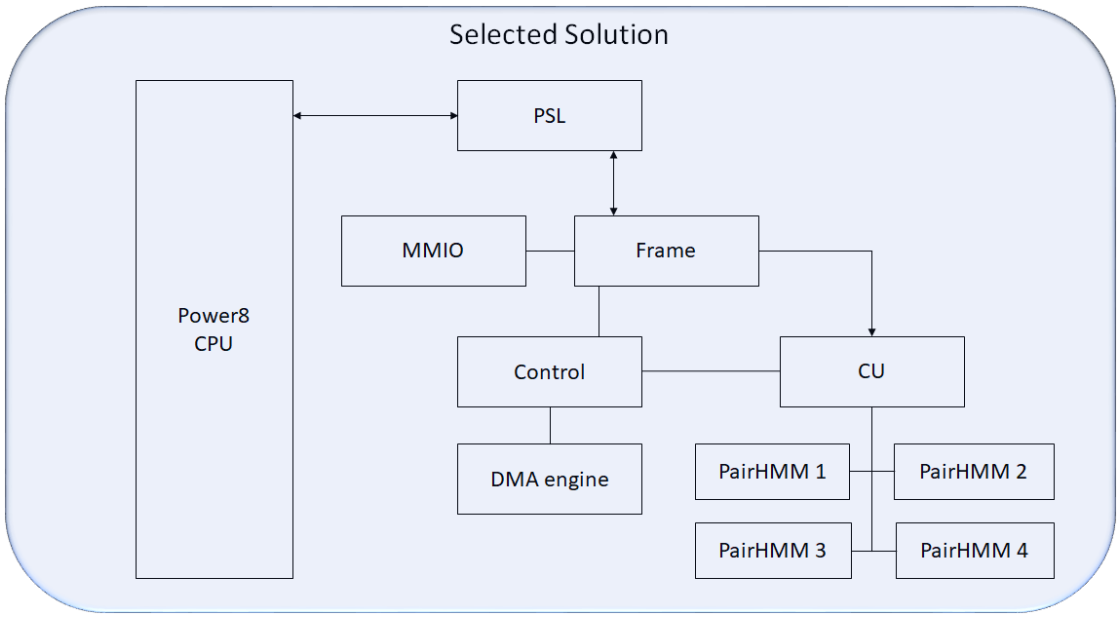


Figure 4.7: Final design

4.2 Modification of HaplotypeCaller

In this section we will describe the modification that should be done in the HaplotypeCaller in order to achieve the integration. At this point, we have the GATK-3 application , an application written in C and our accelerator.

The default C application is responsible for the initialization and execution of the accelerator.

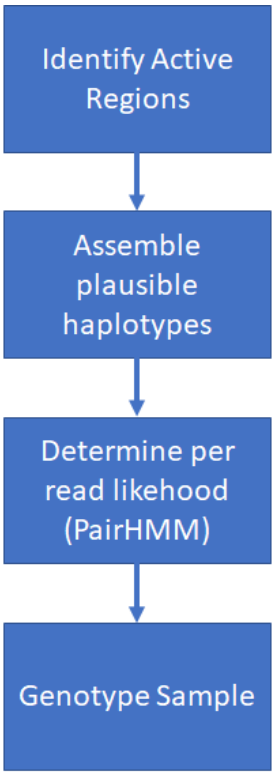


Figure 4.8: Default HaplotypeCaller functionality

Figure 4.8 represents the functionality of the sequential HaplotypeCaller. The application. The first step for HaplotypeCaller is to determine the active regions of genome that will use in the next steps. The identification of the active region is based on the variations of the genome in comparison with the reference sequence.

When the active regions are identified the sequences are sent to the assemble plausible haplotypes step. In this step HaplotypeCaller builds a De Bruijn-like graph to reassemble the ActiveRegion. It also investigates what are the possible haplotypes present in the data. HaplotypeCaller realigns the haplotypes with the reference haplotype by using an DNA alignment algorithm, Smith-Waterman. Smith-Waterman will find the possible variants.

After the execution of Smith-Waterman the chosen data are passed in the third phase of the HaplotypeCaller which is the determination per read likelihoods. The HaplotypeCaller does an alignment for all the reads bases with all the haplotypes. This pairwise alignment is performed by PairHMM algorithm. The outcome of this phase is a matrix of likelihoods of haplotypes given the read data.

Finally, the genotype sample phase is responsible to execute Bayes rule. The most possible genotype in respect with the results of the previous steps will be assigned to the sample.

In this project we modified the third step that the execution of the PairHMM algorithm is being performed in order to replace it with our accelerator. Our target is to collect the data from the phase one and two send them to the accelerator while we will collect the second data set from the same phases. When the results from the accelerator are ready we will pass the results to phase four and we will fetch the new data set to the accelerator.

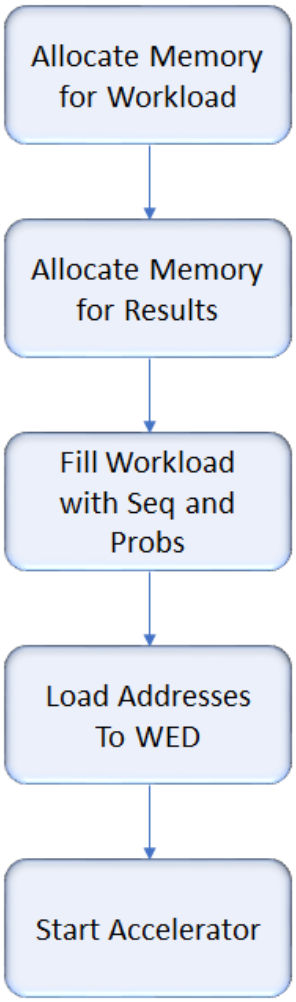


Figure 4.9: Workflow of C application

- Allocate memory for workload.
  The accelerator kernels work for pairs that are multiple of the PEs. The first step is to check how many pairs there are and allocate the correct memory size for the haplotype and reads for the robust function of the system. For that purpose, we make use of the function *load_workload*(). It checks if the pairs are multiple of the PEs and proceed by allocating size of the workload. The next step is to assign the size of each pair and find the maximum per batch. This number will be used in next steps for the correct initialization of the data sets that will accommodate the pairs.
- Allocate memory for the results.
  The next step is to allocate memory for the results of the accelerator. The size of the results vary, for example if the PEs are 16 and the pairs 32 then we have two batches and the result variable has to be capable to accommodate all the results.
- Fill workload.

At this step the actual length of the haplotypes and Reads are known and we are ready to fill the workload with the correct sequences and probabilities. Some extra functions are used in order to pad the sequences when their actual size is smaller than the one that is used for the accelerator. Until this moment the workload is loaded with random data and probabilities that are only for testing purposes.

- Load Addresses to WED
  The data are set in workload and we should initialize the WED with the correct addresses. Those addresses will be used by the accelerator to retrieve the data from the cache and to perform the pairHMM algorithm.
- Mount board and Start the accelerator
  The final step is to mount the device and sent the start signal.

The first two steps are performing calculations to identify the actual size of arrays of data that will be sent to the accelerator. The third step is responsible to load all the final sequences to the arrays and finally we load the different addresses to the WED in order to send all the required information to the accelerator.

4.2.1 Adapted Java part

Before we modify the C application we should change the HaplotypeCaller by adding extra functionality and we should use a method that will allow the communication between Java and C applications. The only method to achieve that is Java Native Interface.

Java Native Interface (JNI) allows us to use any native code when an application cannot be written in the Java language of because we want to increase the performance of the application by using a function in faster language. The following are typical situations where you might decide to use native code. It allows you to use code or code libraries that you want to access from Java programs that are written in different languages. You need platform-dependent features not supported in the standard Java class library.

JNI calls could become really time consuming procedure, so in our approach the first goal is to minimize the number of JNI calls for performance purposes. The solution to this constraint is to use another option that Java API offers sun.misc.Unsafe. Unsafe allows java to perform low level programming such as allocation of memory and data write to specific memory addresses. By combining these two tools that java offers we can allocate memory with the pairs to GATK side and send the memory addresses to C through one JNI call.

In the HaplotypeCaller we store in lists the sizes, sequences, and probabilities for each pair. The second step is to allocate memory through unsafe for read, haplotype,

sizes and probabilities and write to these memory addresses the above data. When we collect the data we are ready to call the native function in C that will perform the accelerators part. The functionality of HaplotypeCaller is described in 4.10.

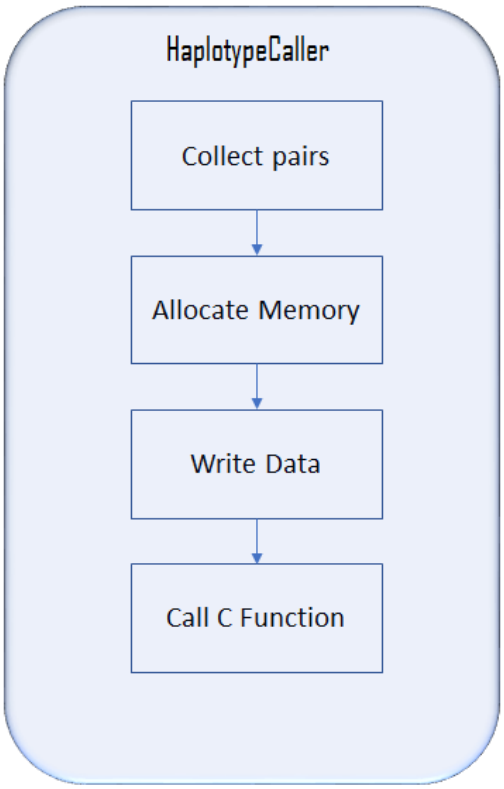


Figure 4.10: Modified HaplotypeCaller

More precisely in the part that PairHMM calculated we replace that segment with the mechanism that will allow the integration. The first part is to receive all the date from the previous step and to store them to our arrays. When we receive them we store them to custom made arrays that will help us to handle them for the integration. The scores that will be used for the calculation of the probabilities are separated and stored to four different arrays because we receive them as a big stream and there are four different score types in this stream.

After we store all the data we are ready to pass them to the C controller of FPGA. For this step we followed different approaches. Firstly, we passed all the arrays through a JNI call. JNI is an function of java that allow to call a C function from java and pass arguments. JNI call with large arrays is expensive in terms of execution and for our data sets it needs more than 100 ms. This communication time will be critical for one of our goals to saturate the FPGA with data sets. After investigation, another API of java discovered sun.misc.unsafe. This API gives the capability to Java to handle memory like

lower programming languages like C. Now it is possible for us to allocate memory and store directly the data. With this procedure our communication time reduced to less 7 ms for our largest data set. The allocation and data storage needs even less time that makes this approach our selection. We allocate memory for the results of the accelerator because that strategy makes the accelerator independent for the HaplotypeCaller. The reason is that when the HaplotypeCaller call the controller and pass the addresses to the controller, then it can start collecting the next data set while tha accelerator calculates the PairHMM. For the new data set we need run again phase one and two of HC. Finally, when the results are ready and the next data set is collected a new run starts for the accelerator.

4.2.2 Adapted C part

The most challenging and intensive part of the integration was the modification of the C controller. The reason that makes it really challenging is that it was created only for testing purposes and for one kernel. Now it should be modified in order to use real data provided from the HaplotypeCaller and validate the preprocess phase that controller should do before we send it to accelerator.

The controller receives a memory address that points to the bases and the probabilities scores for each pair that has been already written in a data line from the HaplotypeCaller. Controller also receives a second memory address with the character size of each base in order to move the first pointer correctly in each iteration. By this way we achieved to handle the data efficient and without meaningless iterations or with the need to re parse the array. Controller in order to retrieve data uses a retrieval mechanism that get as an input the number of pair that we want to retrieve and it calculates the memory address of the exact pair. The way it calculates the memory location is by adding the previous pairs sizes. We should keep in mind that in the memory first we store the read bases characters next the haplotype bases characters and finally for each number read character there are four score numbers. So the mechanism is implemented in way that we can retrieve correctly what we want.

The first problem was faced was that the pairs the 90% of the time was not a multiple of the PEs. So, we implement a padding method to increase the pairs with blanks in order to reach the correct number of pairs that are required for the correct functionality of the accelerator. Another issue is that an unbalanced distribution to the different cores could drive to worse performance than the baseline model. Because the next kernel need some extra from the previous one in order to request and receive the data from the memory before it starts the calculations. If we overload the last kernel and the prior kernels having a really small loads then in the worst case scenario the accelerator will run for the same time as the baseline plus the initialization of the previous kernels.

This problem needs an efficient mechanism that will balance the total number of pairs efficient to the different cores. The other benefit of our design is that when a

result is ready can be used directly by the application while the accelerator calculating the rest pairs in the different cores.

The method that was followed to distribute the pairs is to divide the total number of the pairs by four that represent the number of kernels. This division will define the load of each kernel and if the data cannot be balanced equally to the different kernels then the biggest load will go to the first kernels. Before we call the the load_workload function that calculating the sizes of the data structures that will accommodate the data, we define the number of the pairs for each kernel in respect with the functional requirements.

The workload_function() was modified as well. The workoload_function() is responsible for the correct calculation of the sizes that will be used later on to allocate memory for the data that accelerator needs. We increased the performance of the workload_function() by modifying the algorithm by eliminating a second loop that it was not necessary.

The above function is called four times and in each of them the load balancer manages the inputs of each function in order to achieve the balanced distribution and take advantage of the parallel execution of the four cores of the accelerator. The next step is to load the data to the data structures with the correct structure in order to access them the hardware from the defined address and execute the PairHMM algorithm on them. The function that fills the workload with correct data is called $fill\_batch()$. The first modification in this function was the calculation of probabilities for PairHMM because until now the application provided random values as probabilities. So, we implement the algorithm for the correct calculation of the probabilities as are necessary for the PairHMM calculation. Then we assigned the haplotypes and reads to the correct fields. This procedure is done four times with different data sets by distributing the data to the different with goal the fastest possible performance of the overall system.

More precise the first part of the C controller is to calculate the sizes for the specific data set depending on their sizes. These sizes are stored in a struct that is called workload and we store the size of batch (another data structure that accommodate the PairHMM data) and of the results. The workload was creating until now for random data with fixed sizes. Now that we want to integrate HaplotypeCaller with the new acceleration the controller should be modified for real data.

Workload sizes calculated for one core but now we should modify it in a way that can distribute the data to the four different cores. To achieve that we should check how many pairs exists in the data set and if they are multiple of the PEs. For this purpose we introduced a new mechanism "load balancer".

Load balancer divide the number of pairs by the number four that represents the number of cores. If the number is not a multiple of 16 (PEs) then it will introduce a padding mechanism. The load balancer receives the memory address that contains all the pairs and calls four times the workload function. Because the array contains all the

data we calculate on the fly the new position of the pointer in order no to waste time for no needed parsing of the array.

At this point c controller with load balancer can allocate memory for four cores and only for the actual data even if they are not multiples of PEs. When we have all the sizes in the workloads we will allocate four different aligned memories for the batches. The batch is the data structure that is divided in three parts the reads sequences, haplotypes sequences and the probabilities that are necessary for the PairHMM execution. We allocate 4 different batch structures one for each core and now we are ready to distribute the pairs to them.

If the pairs are 65 and the PEs for each core are 16 then one pair doesn't fit to our batches. For this reason the baseline controller creates a linked list of batches and will run twice the to the core in order to calculate all the pairs. In our solution we created a mechanism that will call distributor and is responsible for the efficient creation of the batches. if we should use 5 batches then the distributor will put two batches on the first core and 1 to the others. This strategy is followed because first core will start earlier and it has more time to operate in comparison with the rests.

Because the cores can work with pairs equals to the PEs the distributor is responsible to add blanks pairs in the remaining positions of the batches. Finally, the batches are loaded to the work element descriptor(WED) that is the data structure that the accelerator can parse. We add a new field on the weds so we can create a linked list of WEDs. This modification allows us to send the address of the first WED and the accelerator will get the others. Another reason that we did this modification is for scalability reasons. If we could increase the number of cores the C controller with the introduced load balancer and distributor can accommodate the new data for the extra core by switching the number of the cores.

Finally, everything is set so we have to allocate memory for four different variables that will be used by the accelerator to store the results of each kernel. C will mount now the board and send a signal to start the execution and will exit. At this point we should explain how HaplotypeCaller will get informed that there are results. For this purpose, was used two different address that were sent to C. The one is used to store the address of the four WEDs. The first 8 bit in each WED represent the status of the kernel, if the status is equal to one then the results are read. The results address were sent to HaplotypeCaller by the same way as WEDs are. Last but not least, each time that the accelerator is finished the calculations in all kernels we should free the FPGA in order to use later on for another data set. In figure 4.11 can are explained the main changes of the baseline C code.

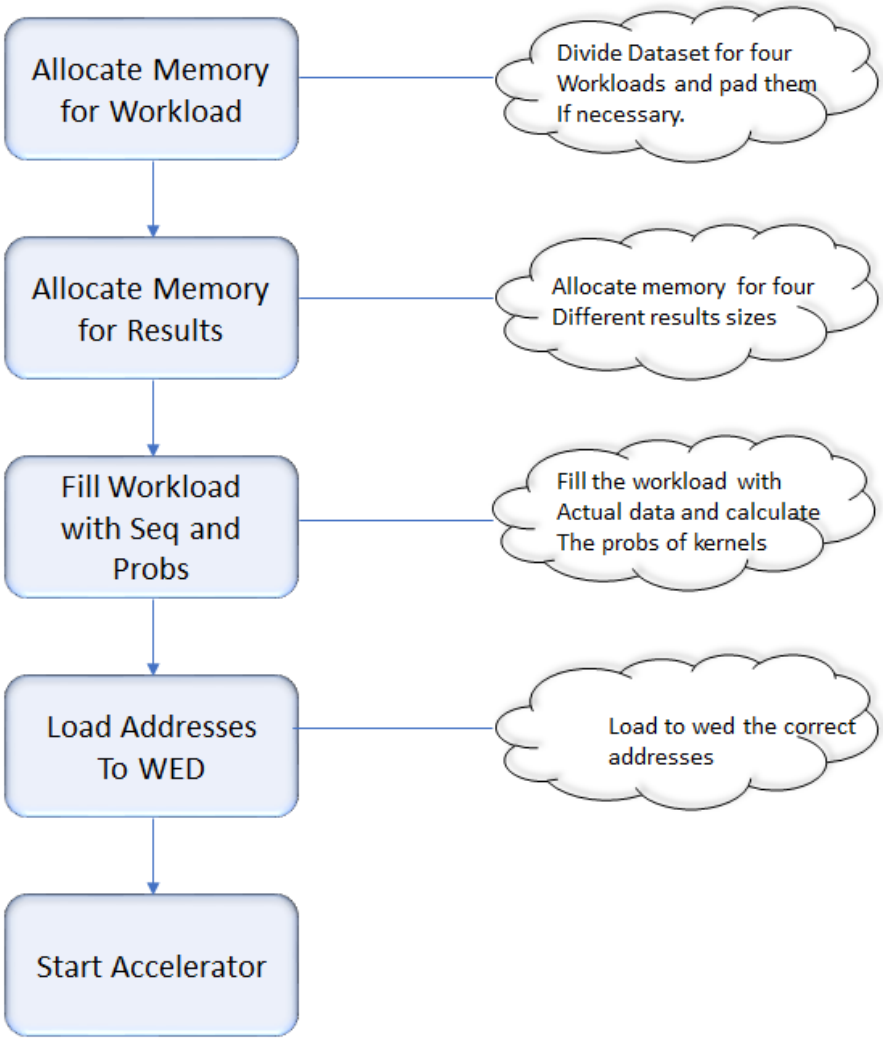


Figure 4.11: Overview of C code

To conclude, we investigate different architecture solutions for the accelerator. The first one was with different DMA engines that each of them would assigned to a kernel. This approach was functional but not efficient as the DMA can service only one engine per cycle and as a result we will have a timing overhead. We could achieve the same functionality by using only one DMA engine and reducing significantly the area of the design. Finally, we met problems to the synthesis step and we used different procedures and we couldn't have a synthesized design.

The overall design with the different layers that used is described in Figure 5.1. The HaplotypeCaller writes all the data in a data line through misc.unsafe. Then it calls the controller by using a JNI call and send the data address of the data. The controller collects and allocates aligned memory for the different types of data that required by the accelerator and finally it sends a start signal and the starting memory address that the

accelerator will reach the data. Finally, while the accelerator calculates the PairHMM algorithm the HaplotypeCaller prepares the next data set.

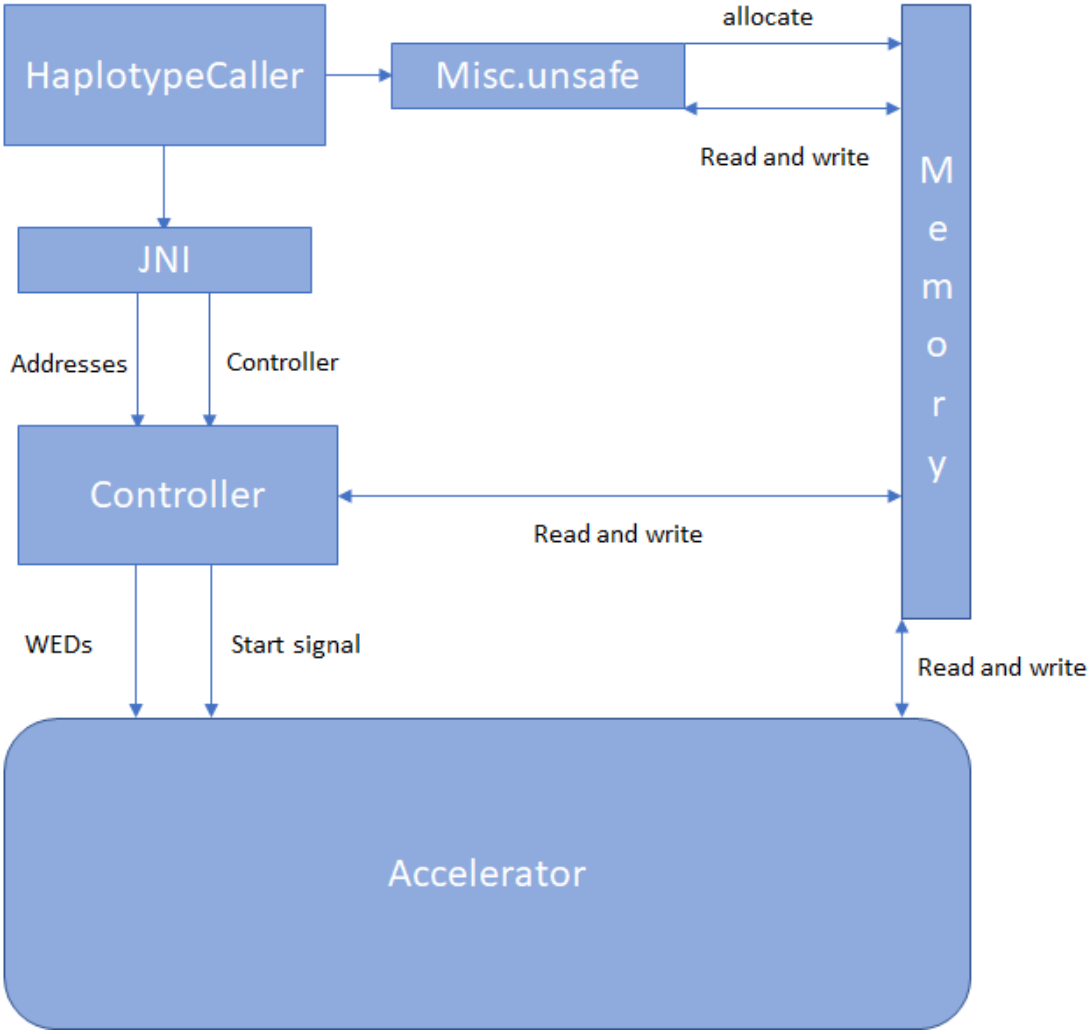


Figure 4.12: Final design

5 Implementation results

In this chapter, the performance of the integration will be presented and discussed. Firstly, we will present the timing results of the accelerator, and we will proceed with the performance of the GATK and of the C controller that enables the accelerator.

5.1 Expected theoretical performance

In this project, we succeeded to design a multicore accelerator for PairHMM with CAPI interface. However, this project couldn't get synthesized because of a timing violation that took place in the PSL interface of CAPI that is provided to us as a block of hardware IP from IBM. The IP block lets us use the design without exposing the source code and without giving us the ability to modify it.

The maximum theoretical speedup that could be achieved from the new accelerator with comparison with the previous one is $4x$. However, after thorough investigation of the design and its functionality, we conclude that each kernel will start calculating after the previous one reaches the 15% of its execution. As a result, each batch loader need 15% of the total execution time, while at the same time we cannot activate all the batch loaders together. In this project, we make use of 3 extra batch loaders and kernels. So the overall overhead with respect to the theoretical performance is 15% * 3 = 45%. Our theoretical speedup will decrease to 4*0.55= 2.2x

In addition we should calculate the maximum bandwidth between the host and the accelerator. The accelerator is connected with a PCIe port that can offer bandwidth of 16GB/s. It is critical to identify the bandwidth when CAPI is going to read or write data through PSL. This bandwidth could provide us a good estimation for the maximum possible saturation of the device. IBM measured read bandwidth when data resides in system memory is 3.68 GB/s and the write bandwidth when data is pushed to system memory is 4.17 GB/s (3.88 GiB/s). The average read latency is 864 ns from PSL request to response. The write latency is 838 ns average from PSL request to response. Latency for both Reads and Writes with PSL cache (AFU's local cache) hit is 120n. We can estimate at this moment that the accelerator is fast enough to handle different dataset from the software part. Is the software capable to prepare the data set in that time segment.

The measurements of the accelerator are presented in Figure 5.1. These measurements produced by the simulated version of the accelerator. We used the timings of modelsim because as we discussed in previous chapter, we could not synthesize the design because of the IBM PSL that occur a timing violation.

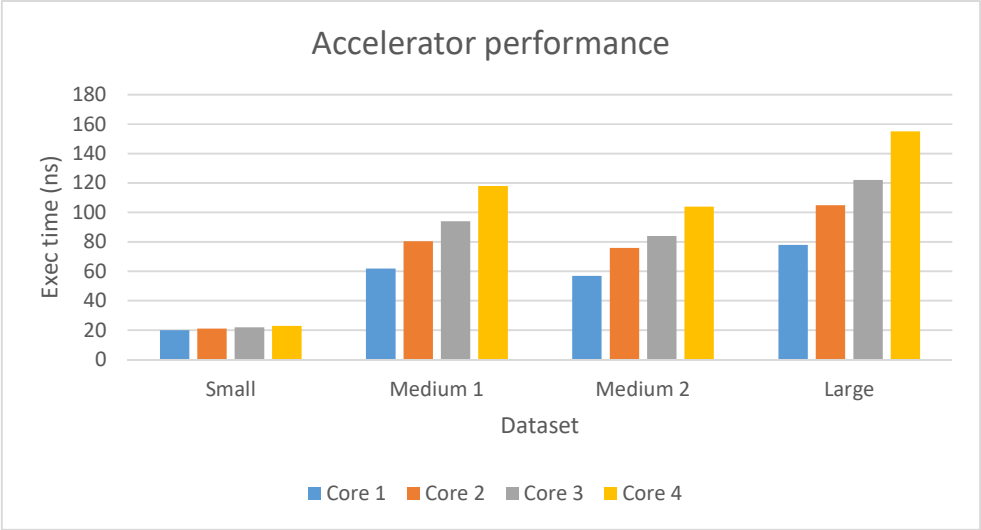


Figure 5.1: Accelerator performance for different datasets

The performance of the accelerator is close to linear. We can observe for each data set in which time segment each kernel is finished because at the time that a kernel is done the results are available to the host side and can be used immediately by the host as it waits for the rest or the kernels to finish.

In terms of throughput each core can achieve the same throughput with the baseline core with the difference that in this design after an initialization period all the cores run in parallel. Ideally our throughput. The new accelerator has four cores each of them contains 16 PEs. The frequency of each core is 166MHz. The maximum theoretical throughput of the accelerator is 16*4*166=10624 MCUPs/s.

However the experimental throughput of this design is approximately the half of the maximum theoretical. The reason is the the different loads of the datasets from the memory. Figure 5.2 presents the throughput of the design in comparison with the baseline design and the maximum theoretical throughput for different numbers of PEs.

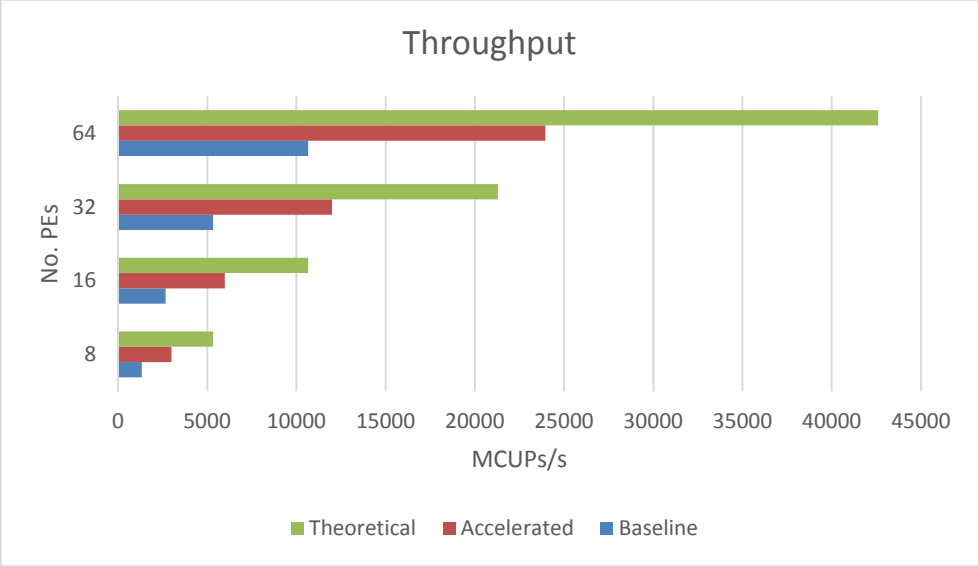


Figure 5.2: Throughput results

5.2 HaplotypeCaller performance

The HaplotypeCaller performance will be presented in this section. We will provide the results of each step for different data sets as well as the overall time that haplotype needs for initialization and calling the accelerator. The first part will be the execution time that it is needed to create the data structures and to reset all the variable for a new data set. Afterward, it is crucial to discuss the time to get Reads bases, haplotypes bases, and the probabilities. At the moment that we would have the above data we will be ready to measure the time for memory allocation, memory writes of the above and finally the time to free up the memory.

For the experiments we used 5 different datasets. The small one with 1322 characters and a total of 20 pairs. Medium 1 with 7762 characters and 144 pairs. Medium 2 with 15624 characters and 80 pairs. Large with 19398 characters and 264 pairs and finally the extra large with over 2 million characters and 14416 pairs.

- Small: 20 pairs of 1322 characters
- Medium 1: 144 pairs of 7762 characters
- Medium 2: 80 pairs of 15624 characters

- Large: 264 pairs of 19398 characters
- Extra large: 14416 pairs of 2162485 characters

The total character number for each dataset is important factor for the different phases and it determines the execution time of each phase. More characters means more calculations. In addition, the different numbers of pairs will show us how the load_balancer reacts.

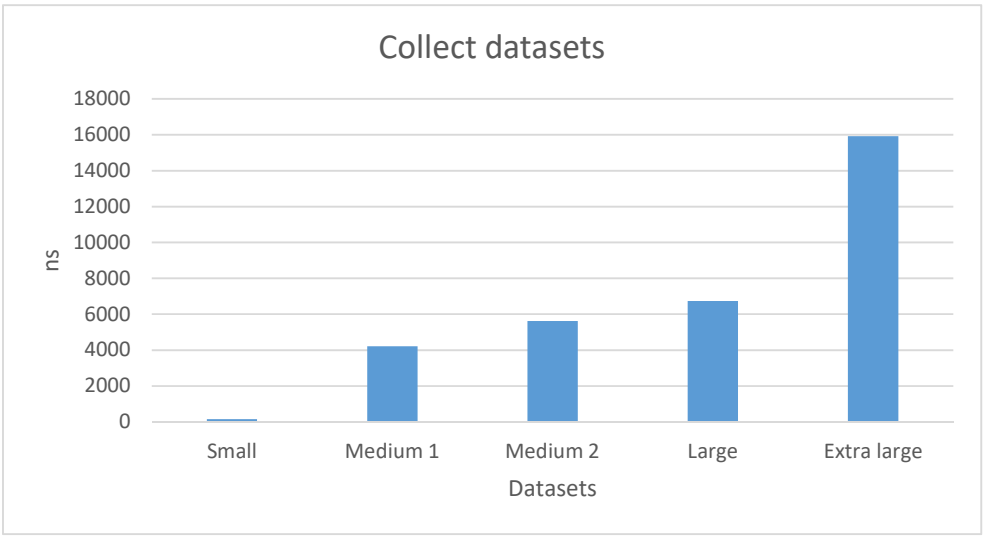


Figure 5.3: Execution time for initializing the data structures and variables

Figure 5.3 shows that the initialization time of the new data structures is dependent on the their size. This is true because we use four data structures in order to store all the values used by the accelerator. When the next data set arrives, we have to re-size these data structures according to the new dataset sizes.

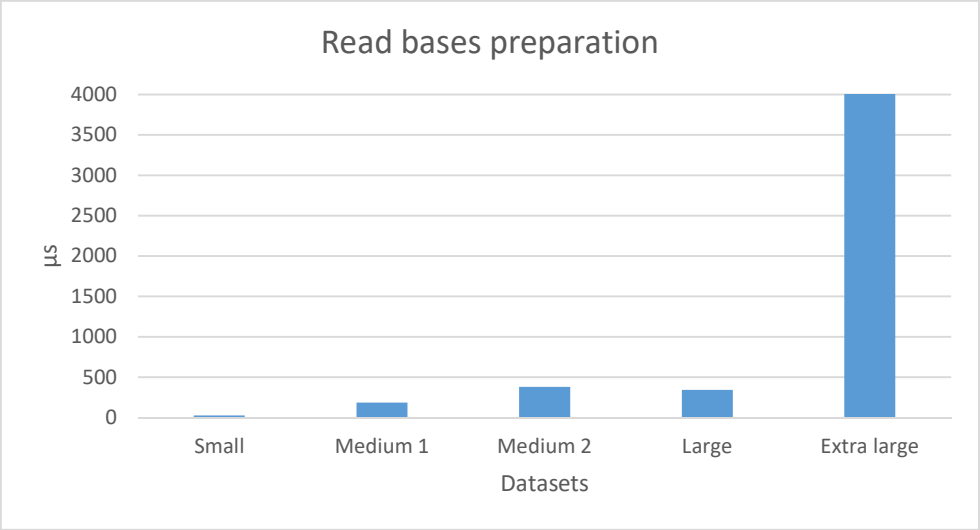


Figure 5.4: Execution time for all reads of a data set

In Figure 5.4, we can see the execution time that the HaplotypeCaller consumes in order to provide all the read bases for a data set. The execution time of the different data sets are the expected one. As the size of the dataset is getting bigger the more time needed from the system. It is interesting to observe that a data set with fewer pairs needs more time than another with a higher number of pairs. This is happening because the number of characters add complexity to the PairHMM cores and not the number of pairs. For example, two pairs of 10 characters each will need less time to be assigned than one of 70 characters length. This behaviour was expected as the calculations becomes heavier if we increase the sequence length.

In haplotype bases assignment we experienced the same behaviour. However we can identify that haplotype assignments are performed faster than the read. This is an outcome of the characters. Haplotypes have smaller number of characters in comparison with reads. This drives to smaller allocation of memory and reduced number of loops for the initialization of the data structures. The results are presented in Figure 5.5.

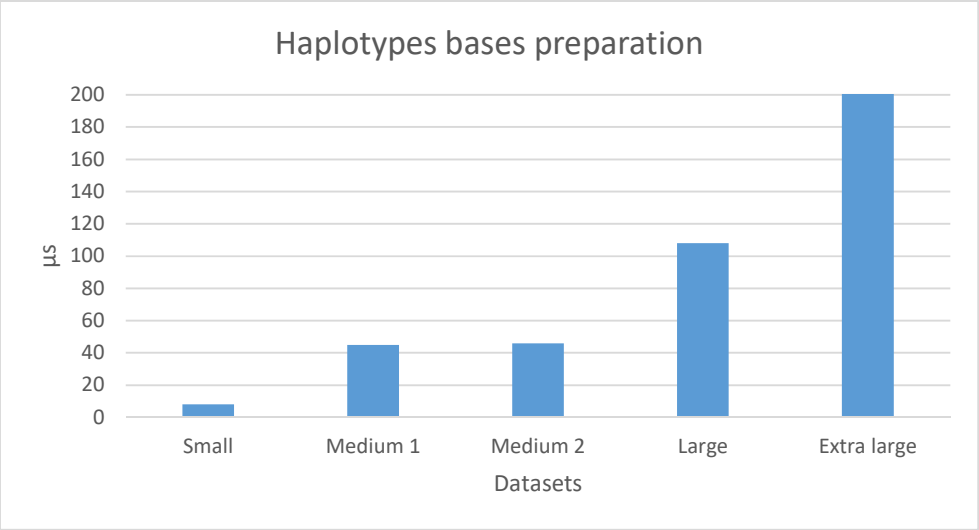


Figure 5.5: Execution time for all haplotypes of a data set

In order to have a functional algorithm of PairHMM, we need some probabilities. For each character of read bases, there are four different probabilities in order to calculate correctly the next state in PairHMM algorithm. This indicates that the time for getting all the probabilities should approximately be four times larger than the reads time. Figure 5.6 shows the execution time of the the probability assignment and it indeed verifies this expectation.

The execution time for collecting the probabilities is not exactly four times larger than the time of the read bases. This is happening because of the different data types. For example, the read bases are a collection of characters which are 8-bit in size in comparison with the probabilities that are integers. The actual size of an integer is 32 bits. This is a fact that will play a crucial role in the part of the haplotype that we should write the above data to specific memory addresses. So we are expecting higher execution time for probabilities again because of the aforementioned reason.

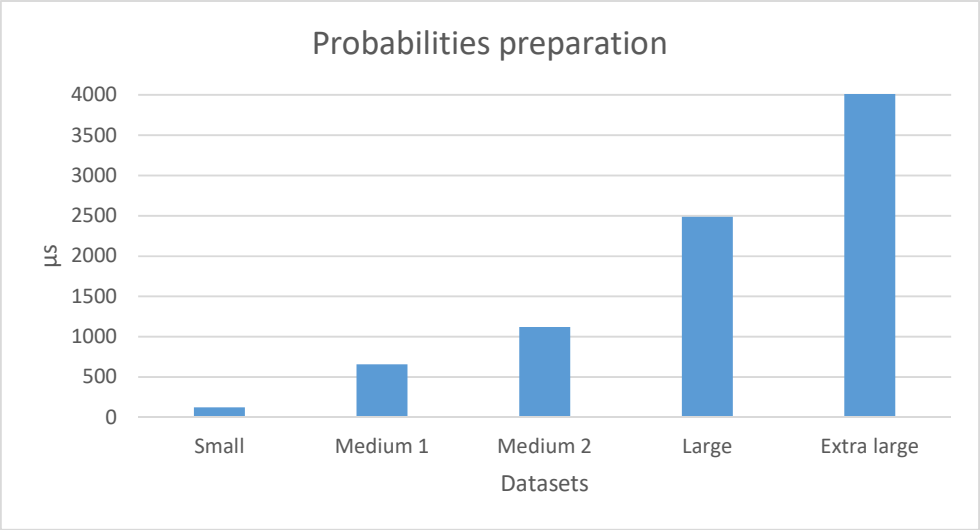


Figure 5.6: Execution time for all probabilities of a data set

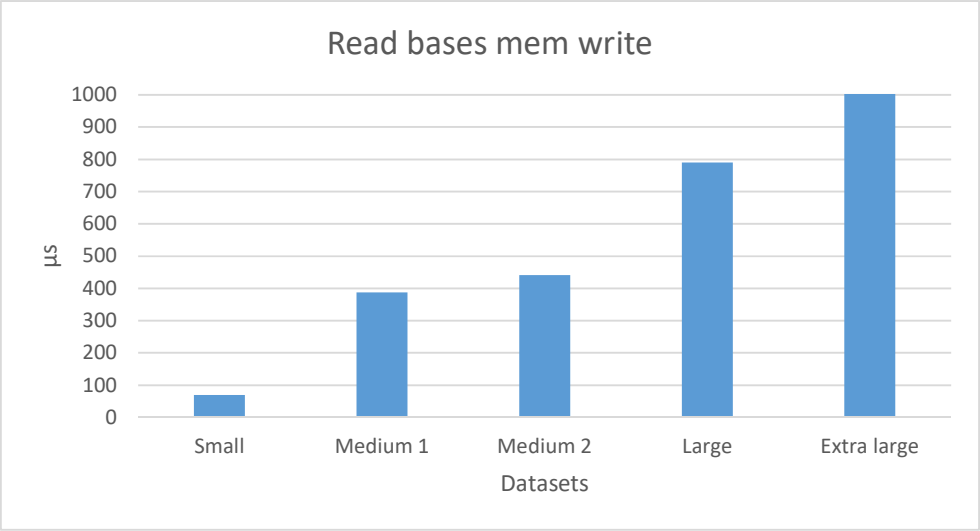


Figure 5.7: Execution time for memory write of reads bases

As it can be seen in Figure 5.7, writing of the read bases in a specific memory address could be a time-consuming procedure. For 80 pairs the execution time is 0.004s which mean that this procedure is expensive in terms of performance and could be the major drawback in the overall performance of the system. As we can see in Figure 5.8 and Figure 5.9, writing the haplotype bases and probabilities is also a time-consuming procedure. We investigate all those execution timings because our target is to identify if the accelerator will finish the calculation before the new data set is ready for execution.

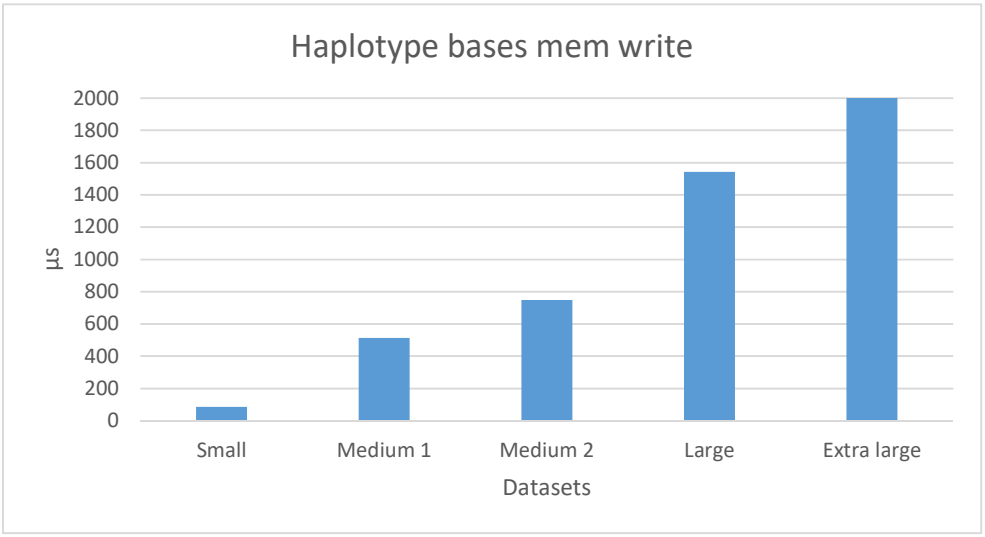


Figure 5.8: Execution time for memory write of haplotype bases

Table 5.1: Overall execution time of the HaplotypeCaller for various number of pairs

Pairs	80	114	20	264	14416
Time in ms	8.542	8.751	0.711	14.565	1445.486

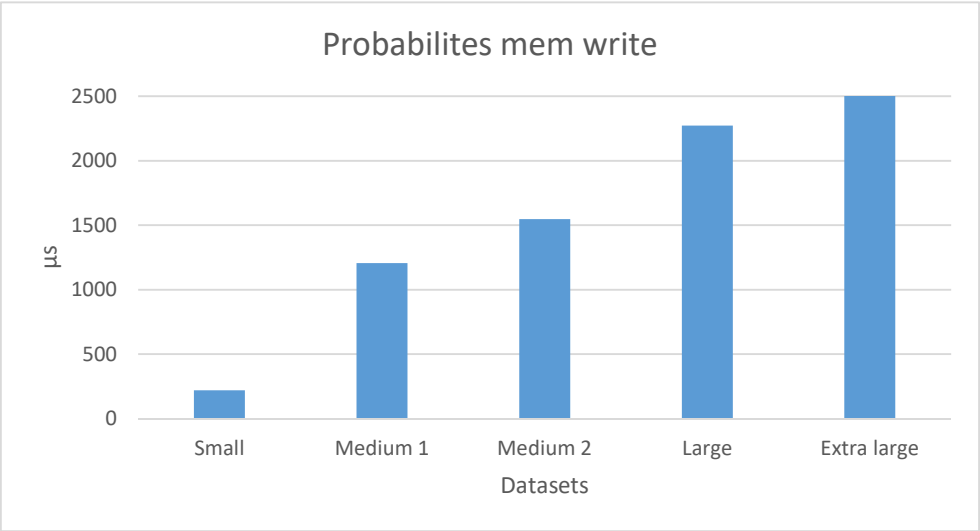


Figure 5.9: Execution time for memory write of probabilities

The final results that we will present are the overall timing that the HaplotypeCaller spends for all the above procedures in total. This will be the main factor that will determine if the application could saturate the accelerator in to order to achieve 100% utilization.

Table 5.1 contains the overall execution time of the HaplotypeCaller for a different number of pairs. We can observe that for 80 pairs and 114 the overall execution is quite the same. This is a result of different length of haplotypes and reads which are really important for the execution time. However, these results represent only 50% of the results of the integration. In the next section, we will present the time needed in the C part of the integration. Both timings are really important in order to check if the host side can saturate the accelerator and if not which part of the code is the most time-consuming.

5.3 C performance

In this section, we will present the times that the C part needs in order to operate correctly and to load the accelerator correctly. This part of the integration will always be executed when the accelerator has finished the previous calculations. In other words, this part of the code cannot be calculated in parallel with the accelerator because the C part not only carries out the initialization, it also manages the manages its operation.

Table 5.2: Execution times the C application part needs to operate correctly, times in milliseconds

Pairs	80	114	20	264	14416
Workload	0.216	0.246	0.113	0.375	13.766
Fill batches	0.716	0.646	0.213	2.375	142.766
Results Allocation	0.018	0.018	0.020	0.020	0.021
Total	0.958	0.874	0.387	2.965	155.924

In Table 5.2 all the times are in millisecond. We can observe that for a low number of pairs the overall execution time is low. For a larger number of pairs like 14416, the overall time of the C application increases accordingly to about 155x time than that for 80 pairs. Another crucial part to observe is that the most intensive part of the code is the part that fills the batches and could be the 90% of the overall execution time on some occasions.

6 Conclusion and future work

The goal of this project was to create an accelerator with multiple independent kernels of PairHMM algorithm for high performance calculations while maintaining the accuracy of the system. The designed accelerator was adapted to the HaplotypeCaller with success using $misc.unsafe$ in order to allocate and write to the memory all the needed data for the accelerator. After the successful write of data one JNI call is used to send the memory addressed to a C application that will prepare and send the probabilities and the sequences to the accelerator.

The new design is more efficient than the previous one. The maximum theoretical speedup that we could achieve was 4x which is higher than the actual 2.2x speedup that we achieved. This resulted because of the design of the DMA that can serve only one request per cycle. This creates a communication overhead for the kernel since it limits the amount of data each kernel can receive at each point of time, something that adds 20% extra execution time.

The performance of the new accelerator for different data sets can be seen in Table 6.1. Each kernel needs 20% more execution time for the previous one and the maximum speedup is almost 2.2x. The achieved throughput is 5857,2 MCUPs/s for 16 PEs

The host side is modified and we tried to find the most efficient way to achieve the communication between software and accelerator. The application is responsible for preparing the pairs and loading them to the accelerator. For a small number of pairs, the execution time is close to 8ms, while for a large number of pairs (e.g., 14400) this number is more than 1400ms. Now, we should figure out if the software could saturate the accelerator.

Saturating the accelerator is the responsibility of the host side of the system, which prepares and loads a new data set while the accelerator is still calculating the previous one. If the accelerator executes a small number of pairs, while the host is simultaneously preparing a large one, then the accelerator should wait in idle mode. If the opposite

Table 6.1: Performance of the new accelerator for different data sets

Data sets	Small	Medium 1	Medium 2	Large
Core 1	26ms	58ms	67ms	81ms
Core 2	27ms	74ms	80ms	110ms
Core 3	28ms	87ms	93ms	130ms
Core 4	29ms	107ms	115ms	154ms

happens, then the accelerator will work without going to idle mode at all. From the data that was used in this project, 26% of the overall pairs used force the accelerate to wait. A better approach of loading and preparing the data to the accelerator could help to decrease this percentage of idle time.

For future development, it would be beneficial to design our own PSL that we could adapt in respect to each project to achieve even higher performance and to diminish all the timing violations that occurred by the PSL (as it was provided by IBM). The host side should be investigated for modification in terms of higher performance; this is the only way to achieve a saturation of the accelerator.

Finally, it is important to evaluate the proposed scheduling in the C part of the program that is responsible for the distribution of the different pairs per kernel. This could help us find out if there is a better scheduling approach. At the same time in HaplotypeCaller, it should be identified if the load of four data sets is more beneficial than the load a single data set to the four kernels.

To sum up, a hybrid system for our project could be beneficial as the collaboration of software with a hardware accelerator offers 2.2x speedup in comparison with the baseline. The application is not fully capable of saturating the FPGA card, which results in introducing some idle time during execution. Some extra investigation and optimization of the software can diminish the idle time of the FPGA. This will help improve the overall performance of the application since our implemented multicore accelerator with four cores of PairHMM is able to offer higher throughput of up to almost four time that of the single core design. This means that it will not be a bottleneck if the application is able to provide input data to process at a higher rate.

Bibliography

[1] A. H. Paterson, S. D. Tanksley, and M. E. Sorrells, "{DNA} markers in plant improvement," ser. Advances in Agronomy, D. L. Sparks, Ed. Academic Press, 1991, vol. 46, pp. 39 – 90. [Online]. Available: //www.sciencedirect.com/science/article/pii/S0065211308605787

[2] "http://www.sciencemag.org/news/2017/03/dna-could-store-all-worlds-data-one-room," Retrieved February 2017.

[3] "Dna: Definition, structure & discovery, rachael rettner," 2013. [Online]. Available: http://www.livescience.com/37247-dna.html

[4] "Modern applications of dna. boundless biology boundless," 2016. [Online]. Available: https://www.boundless.com/biology/textbooks/boundless-biology-textbook/dna-structure-and-function-14/historical-basis-of-modern-understanding-99/modern-applications-of-dna-432-11660/

[5] "http://www.slideshare.net/avrilcoghlan/the-needleman-wunsch-algorithm," Retrieved January 2017.

[6] "https://web.stanford.edu/class/cs262/archives/notes/lecture8.pdf," Retrieved January 2017.

[7] "http://vlab.amrita.edu," Retrieved January 2017.

[8] "http://www.networks.howard.edu/chunmei/cb/ch4-alignment.ppt," Retrieved January 2017.

[9] "https://software.broadinstitute.org/gatk/," Retrieved January 2017.

[10] H. Mushtaq, Z. Al-Ars, and P. Hofstee, "Sparkga: A spark framework for cost effective, fast and accurate dna analysis at scale," in *Proc. 8th ACM Conference on Bioinformatics, Computational Biology and Health Informatics*, Boston, USA, August 2017.

[11] *Computational Challenges of Next Generation Sequencing Pipelines Using Heterogeneous Systems*, Fiuggi, Italy, July 2016.

[12] *Cluster-based Apache Spark implementation of the GATK DNA analysis pipeline*, Washington DC, USA, November 2015.

[13] E. Houtgast, V. Sima, G. Marchiori, K. Bertels, and Z. Al-Ars, "Power-efficiency analysis of accelerated bwa-mem implementations on heterogeneous computing platforms," in *Proc. International Conference on Reconfigurable Computing and FPGAs*, Cancun, Mexico, December 2016.

[14] E. Houtgast, V. Sima, K. Bertels, and Z. Al-Ars, "An efficient gpu-accelerated implementation of genomic short read mapping with bwa-mem," in *Proc. International Symposium on Highly-Efficient Accelerators and Reconfigurable Technologies*, Hong Kong, China, July 2016.

[15] E. Houtgast, V. Sima, G. Marchiori, K. Bertels, and Z. Al-Ars, "Power-efficient accelerated genomic short read mapping on heterogeneous computing platforms," in *Proc. 24th IEEE International Symposium on Field-Programmable Custom Computing Machines*, Washington DC, USA, May 2016.

[16] E. Houtgast, V. Sima, K. Bertels, and Z. Al-Ars, "Gpu-accelerated bwa-mem genomic mapping algorithm using adaptive load balancing," in *Proc. 29th International Conference on Architecture of Computing Systems*, Nuremberg, Germany, April 2016, pp. 130–142.

[17] N. Ahmed, V. Sima, E. Houtgast, K. Bertels, and Z. Al-Ars, "Heterogeneous hardware/software acceleration of the bwa-mem dna alignment algorithm," in *Proc. International Conference On Computer Aided Design*, Austin, USA, November 2015, pp. 240–246.

[18] E. Houtgast, V. Sima, K. Bertels, and Z. Al-Ars, "An fpga-based systolic array to accelerate the bwa-mem genomic mapping algorithm," in *Proc. International Conference On Embedded Computer Systems: Architectures, Modeling, And Simulation*, Samos, Greece, September 2015.

[19] N. Ahmed, K. Bertels, and Z. Al-Ars, "A comparison of seed-and-extend techniques in modern dna read alignment algorithms," in *Proc. Workshop on Accelerator-Enabled Algorithms and Applications in Bioinformatics*, Shenzhen, China, December 2016, pp. 1426–1433.

[20] S. Ren, K. Bertels, and Z. Al-Ars, "Exploration of alternative gpu implementations of the pair-hmms forward algorithm," in *Proc. 3rd International Workshop on High Performance Computing on Bioinformatics*, Shenzhen, China, December 2016.

[21] S. Ren, V. Sima, and Z. Al-Ars, "Fpga acceleration of the pair-hmms forward algorithm for dna sequence analysis," in *Proc. International Workshop on High Performance Computing on Bioinformatics*, Washington DC, USA, November 2015, p. 6.

[22] H. T. Kung and C. E. Leiserson, "Systolic arrays for vlsi, interim report, department of computer science, carnegie mellon university," in *Proc. 3rd International Workshop on High Performance Computing on Bioinformatics*, 1978.

[23] J. Peltenburg, S. Ren, and Z. Al-Ars, "Maximizing systolic array efficiency to accelerate the pairhmm forward algorithm."

[24] B. Wile, "Coherent accelerator processor interface (capi) for power8 systems."

[25] "Coherent accelerator processor interface (capi) for power8 systems white paper," Retrieved January 2017.

[26] "Reconfigurable accelerators for big data and cloud raw 2016," Retrieved January 2017.

[27] T. P. Morgan, "Opening up the server bus for coherent acceleration," October 17, 2016.

[28] M. Ito and M. Ohara, "A power-efficient fpga accelerator: Systolic array with cache-coherent interface for pair-hmm algorithm," 2016.

[29] Xilinx, "Vivado design suite user guide," April 2 2014.