INDIAN INSTITUTE OF TECHNOLOGY DELHI
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

REPORT OF THE PROJECT AWARDED UNDER SURA -2011

Project titled: **FPGAs as accelerators for Next-Generation Sequencing Applications**

Submitted by:

Name of the Student	Deptt./Center	Entry No.	Contact No.	Signature of the student
Dhruv Jain	Dept. of Computer Scince and Engg.	2009CS10187	9873246201	

Name and Signature of the Facilitator:

Prof. M. Balakrishnan

His/her Deptt./Centre: Dept. of Computer Science and Engg.

Mobile No. of the Facilitator: +919871666611

Data of Submission of Report in IRD:

Abstract

Bioinformatics is a field which concerns with application of information technology and computer science to the field of molecular biology. A very popular discipline in bioinformatics is Next-generation sequencing or DNA sequencing. It specifies sequencing methods for determining the order of nucleotide bases-adenine, guanine, cytosine, and thymine in a molecule of DNA which is then assembled for analysis. A central challenge in DNA sequencing is sequence alignment, whereby fragments of much longer DNA sequence are aligned and merged in order to construct the original sequence. A wide variety of alignment algorithms have been subsequently developed over the past few years. Some commonly used softwares implementing these algorithms are BWA-SW, Bowtie, Velvet, SOAP and MAQ. Most of these take lot of time to execute on general purpose processors. Hardware accelerators such as FPGAs and GPUs can be used with processors to fasten these applications. As per our knowledge there are very few reports published in literature about accelerating these applications using FPGAs. In this report, we first exploit the performance and memory issues in processor based implementation of various algorithms and choose one out of them for implementation. Secondly, we present the design of the FPGA based implementation of the chosen software tool via Hardware-Software codesign to achieve a considerable increase in performance as compared to its processor implementation.

Introduction

In bioinformatics, sequence alignment is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a conse-quence of functional, structural, or evolutionary relationships between the sequences. In other words, it is a scheme of writing one sequence on top of another where the residues in one position are deemed to have a common evolutionary origin.

NGS or DNA Sequencing is a part of sequence alignment whereby, nu cleotide bases adenine, guanine, cytosine and thymine are aligned against a reference genome for assembly at a later stage. The fragments of DNA are matched against a given genome using some algorithm so that it can be assembled at a later stage for DNA analysis.

Next Generation Sequencing Problems and Complexity

Knowledge of DNA sequences has become indispensable for basic biological research. It is also used in other research branches which utilize DNA sequencing. Numerous applied fields such as diagnostic, biotechnology, forensic biology and biological systematics have come up which use DNA sequencing for their applications. The advent of this technology has significantly accelerated biological research and discovery[17][18].

Genomes vary widely in size. The smallest known genome for a free-living organism(a bacterium) contains about 600,000 DNA base pairs, while human and mouse genomes have some 3 billion. Next generation sequencing technology cannot read whole genomes in one go, but rather small pieces between 20 and 1000 bases, depending on the technology used. So there is a need to align those read sequences in order to assemble and construct the whole DNA to be useful for analysis.

Fortunately, projects by scientific collaboration across continents, have generated the complete DNA sequences of many animal, plant, and microbial genomes. The speed of sequencing attained with modern DNA sequencing technology has been instrumental in the sequencing of the human genome, in the Human Genome Project.

Sequence Alignment Algorithms

The first successful gapped sequence alignment algorithm was due to Smith- Waterman[19]. They formulated the alignment problem as a finite optimization problem which was solved by dynamic programming. Although database sizes have increased such that Smith-Waterman is no longer practical to use, it is helpful as a base line by which to measure both the performance and quality of heuristic algorithms. In the past few years many efficient algorithms have come up which align sequences in very little time. The latest ones include BWA-SW[20] , Bowtie[21] , Mosaik, Velvet[22] , SOAP2[23] and MAQ[24].

Burrows- Wheeler Transform (BWT)[25] is an algorithm which focuses on prefix trie which is the trie of the reverse string. Suffix trie, or simply a trie, is a data structure that stores all the su#xes of a string, enabling fast string matching. All algorithms on a trie can be seamlessly applied to the correspond-ing prefix trie. BWT-SW essentially sample sub- strings of the reference by a top-down traversal on the trie and align these substrings against the query by dynamic programming.

Burrows- Wheeler Aligner's Smith-Waterman Alignment (BWA-SW), is used to align long sequences up to 1 Mb against a large sequence database (e.g. the human genome) with a few gigabytes of memory. BWA-SW furthers BWT-SW by representing the query as a directed word graph (DAWG) [26], which also enables it to deploy heuristics to accelerate alignment. The algorithm is as accurate as SSAHA2[28], more accurate than BLAT[27], and is several to tens of times faster than both.

Bowtie is an ultrafast, memory-efficient alignment program for aligning short DNA sequence reads to large genomes. For the human genome, Burrows- Wheeler indexing allows Bowtie to align more than 25 million reads per CPU hour with a memory footprint of approximately 1.3 gigabytes. Bowtie extends previous Burrows-Wheeler techniques with a novel quality-aware backtracking algorithm that permits mismatches. Multiple processor cores can be used simultaneously to achieve even greater alignment speeds.

Velvet manipulates de Bruijn graphs for genomic sequence assembly. A de Bruijn graph is a compact representation based on short words (k-mers) that is ideal for high coverage, very short read (25-50 bp) data sets. Applying Velvet to very short reads and paired-ends information only, one can produce contigs of significant length, up to 50-kb N50 length in simulations of prokaryotic data and 3-kb N50 on simulated mammalian BACs. When applied to real Solexa data sets without read pairs, Velvet generated contigs of about 8 kb in a prokaryote and 2 kb in a mammalian BAC, in close agreement with our simulated results without read-pair significantly improved version of the short oligonucleotide alignment program that both reduces computer memory usage and increases alignment speed at an unprecedented rate. It uses a Burrows Wheeler Transformation (BWT) compression index to substitute the seed strategy for indexing the reference sequence in the main memory. When tested on the whole human genome, it is found that there is reduced memory usage from 14.7 to 5.4 GB and improved alignment speed by 20#30 times. SOAP2 is compatible with both single and paired-end reads. Additionally, this tool now supports multiple text and compressed file formats. A consensus builder has also been developed for consensus assembly and SNP detection from alignment of short reads on a reference genome.

MAQ can build assemblies by mapping shotgun short reads to a reference genome, using quality scores to derive genotype calls of the consensus sequence of a diploid genome, e.g., from a human sample. MAQ makes full use of mate- pair information and estimates the error probability of each read alignment. Error probabilities are also derived for the final genotype calls, using a Bayesian statistical model that incorporates the mapping qualities, error probabilities from the raw sequence quality scores, sampling of the two haplotypes, and an empirical model for correlated errors at a

site. Both read mapping and genotype calling are evaluated on simulated data and real data. MAQ is accurate, efficient, versatile, and user-friendly.

FPGAs as Hardware Accelerators

Hardware accelerators employ the use of hardware which can be coupled with general purpose processors and super computers to perform some task faster than software. Many type of hardware devices are available such as FPGAs and GPUs for many applications. Accelerator technology has become very popular in the community in the past decade. Recently many researchers have reported accelerator implementations for sequence alignment[4][5]. FPGA is a class of hardware accelerators that can be programmed after manufacturing. Instead of being restricted to any predetermined hardware function, an FPGA allows to program product features and functions, adapt to new standards, and reconfigure hardware for specific applications even after the product has been installed in the field hence the name "field-programmable".

Review of Literature

Sequence alignment manifests itselfs in two forms : short and long reads. The term 'short read' came about to describe technologies that produced reads that were substantially shorter (30-50 bp) than the mainstream technologies employed at that point (1000bp). Researches have tried to speed up the sequence alignment algorithms in both fields. A comparison of the achieved speed-up for long read alignment algorithms is shown in the table 1.

Program	100bp	200bp	500bp	1000bp	10000bp
BLAT	559	486	512	599	710
SSAHA2	9345	5252	6863	3112	-
BWA-SW	84	118	152	150	120

Table 1: A comparison of approximate running times of long-read alignment software tools. Approximately 10000000 bp data of different read lengths are simulated from the human genome.
These simulated reads are aligned back to the human genome with BLAT (option -fastMap), BWA-SW and SSAHA2 (option =454 for 100 and 200 bp reads). In each cell in this table, the numbers are the CPU seconds on a single-core of an Intel E5420 2.5 GHz CPU. Note the speedup of BWA-SW algorithm compared to the old ones[8].

Algorithm	Author(s)	Host Processor	FPGA	Speedup
S-W	Li, I. T., Shum, W.,T[6]	2GHz Intel Pentium 4	Stratix EP1S40	160x
S-W	Zhang, Tang, Gao[7]	2.2GHz AMD Opteron	XD1000	200x

Table 2: A comparison of FPGA implementations of Smith-Waterman algorithm for two researches. The speedup is measured with FPGA(and processor) based implementation of the algorithm as compared to pure software implementation. There is a considerable speedup obtained.

For FPGAs, it is very difficult to come up with a comparison of the different implementations, as there is no fixed benchmark for testing the performance. Different authors present different ways to claim that their implementation is best. Here we make an effort to analyze the various implementations. Some speedup results for the hardware-software codesign implementations as compared to pure software implementation is shown in table 2.

There have have been implementations of very few old algorithms like smith-waterman on

FPGAs[4][5] and they have reported a considerable speedup. Besides, Burrows wheeler transform(BWT) algorithm which is a central approach for some of the algorithms has also been implemented on FPGA[1]. Results showed a reduction in more than 40% the number of cycles required to perform the complete task compared with previous solutions and an increase in maximum frequency.
Some hash based algorithms have also shown an increase in performance when implemented on FPGA[2][3]. Since the above mentioned algorithms are based on BWT and hash algorithms, they were also likely to show a speedup when implemented on FPGAs.

Methodology

Here is a breif description of various stages in the project:

- **Profiling**: The C/C++ code of the above open source software tools under GPL licence have been downloaded from the web. Profiling of these is done by running them on real data. The compute intensive kernels for each tool have been identified.

- **High Level Performance Estimation**: We have done the performance estimate of running the kernels in the FPGA. An analysis has been done on these to identify the application best suited for FPGAs. This analysis is based on the time critical kernel. We have consider the rough estimate of the resources consumed by the kernel on the FPGA. From this, we are able to estimate the number of functional units to exploit parallelism. Speedup also depends on the communication interface and memory hierarchy. It has also been modelled at a higher level to estimate the speedup.

- **Selection of desired algorithm**: Based on the performance estimation of each algorithm we have chosen the best application for the final implementation on FPGA.

- **Hardware-Software Codesign**: The performance estimate has been used for doing hardware software codesign where the code is divided into specfic parts which exclusively execute on FPGA and on processor.

Profiling Results

The various possible classifications of sequence alignment algorithms include classification on the basis of completeness viz. gapped(where insertions and deletions are considered) or ungapped (opposite of gapped); on the basis of no. of query sequences sets viz. single-end(where only single set of query sequences are used to align), paired-end(two sets of query sequences each from two different ends are used) or de-novo(where there is no reference gnome) and; on the basis of length of reads viz short(query sequences are short usually <100bp) or long read(about 400bp).

Due to simplicity and completeness, I chose to consider only the gapped, single-end read algorithms.

Apparatus:

- **Tools**
  1. *wgsim[9]*, a utility in the samtools package, for generating the random reads, both short and long reads
  2. *Intel Vtune Amplifier[10]*, for profiling and locating time and memory critical areas.

- **Subject benchmarks:** These include the following reference Human Chromosomes(hg 19) downloaded from UCSC Genome Database[11]
  1. Chromosome 19
  2. Chromosome 10
  3. Chromosome X
- **System:** Sony Vaio (Intel Core i3, Ubuntu10.10, 3GB RAM) was used for the profiling
- **Open Source Softwares:** Bwa, Bwa-sw[12]**,** Maq[13] and Soap[14] were profiled

Observations: Presented in table 3 are some brief results of alignment of various query sequence against reference genome. These results agree with the researches of various authors[15][16].

Chromosome	Size of the reference genome(MB)	Algorithm	No. of bp in the query sequence	Time taken to index(s)	Time taken to align(s)
10	131.8	maq	70	40.4	741.6
		soap	70	194.1	79.4
		bwa*	70	232.4	315.2
		bwa-sw*	500	232.4	8838.5
19	57.5	maq	70	31.5	865.2
		soap	70	78.2	84.2
		bwa	70	92.7	359.4
		bwa-sw	500	92.7	8030.5
X	151	maq	70	41.1	802.7
		soap	70	225.8	86.4
		bwa	70	267.5	342.6
		bwa-sw	400	267.5	11269.8

Table 3: Profiled results of various software tools(in their default configuration) with the given chromosomes in fasta sequences as input. The Index column gives the time taken to generate the files from fasta sequences required for providing input to the alignment tool. The queries were generated thorough wgsim in the fastaq format.

Algorithm	Function	% Contrib	Remarks
bwa-index	bwa_index	100	It calls a lot of functions each consuming a significant amount of time.
bwa	bwa_cal_sa_reg_gap	99.1	bwa_cal_sa_reg_gap is a sub-function of bwa_aln_core which constitutes a critical part of it
	bwa_aln_core	100	
bwa-sw	bsw2_aln_core	99.6	The functions above it constitute a lot of code but take very less time.

Table 4: Time Critical Areas in “Burrows- Wheeler Aligner”. The functions shown are part of the c code obtained under GPL license. The functions(and there sub-functions) are only a rough estimation of the code which will actually be implemented in HDL and onto FPGA. Rest of the code will be implemented on the processor and interfacing would be done.

*bwa is for short read whereas bwa-sw is for long read sequence alignment. Rest are short read

Conclusion: I chose to use bwa for implementation on fpga because:

1. The profiling results as summarised in table 4 show that it is likely to show an increase in performance when implemented on FPGA.
2. The algorithm is recent and highly cited with about 300 citations as of Jun 10 2011. It is one of the most widely used software tool, “Burrows Wheeler Aligner”.

Design of the algorithm

Code Analyses

The code is executed in loops with each iteration processing a single sequence in 0.32 ms time as shown in figure 1. The reference gnome is kept in the memory and the short sequences are aligned with it. For a reference gnome of size 153.3MB which is one of the largest gnome in human body*, the data required is 60MB and for 262,144 sequences of 50bp each taken at a time, the data passed constitutes 20.95MB. The result of all these sequences constitute 5.95MB which will be returned back for output. The local memory requirements within the function is 145MB. Detailed analysis is shown in the table 5.

In each iteration, the query sequences are compared sequentially with short parts of the reference gnome at a time. The results are recorded in order and when the alignment is complete they are returned back to the CPU.

S.No.	Variable	Quantity(in no.)	Memory Usage
1	Reference Gnome	2**	60MB
2	Query sequences	262144	20.95MB
Total(including some extra local variables)			145MB

Table 5: Detailed analysis of local memory requirements. The depicted results are for 153.3MB reference gnome with 262,144 short sequences of 50bp each. Further, the memory required is directly proportional to the size of reference gnome and the number of sequences.

Hardware-Software Codesign

Code Analysis conveyed that memory required for each iteration is almost similar but not exactly the same. So, we decided to use small FIFO buffers for storing a part of the reference gnome to source the FPGA iterative blocks. This is because FIFO buffers can be fed with a small amount of data^ from DRAM at a time and can be used independently. The same data will be passed to the FIFO buffers in fixed intervals of time. The iterative blocks in FPGA will use the data from FIFO buffers sequentially and execute the code. Care would be taken to ensure that none of the buffers become empty. The query sequences which are smaller in size will be directly passed to the registers of the iterative blocks. After processing, the results will be returned to the CPU via the same DRAM. The entire process is shown in Figure 2.

* according to human gnome project, chromosome 10 of hg19 database
** Another copy of reference gnome is also stored
^ Reference gnome is quite large. Making multiple copies of it may result in running out of limited FPGA memory.

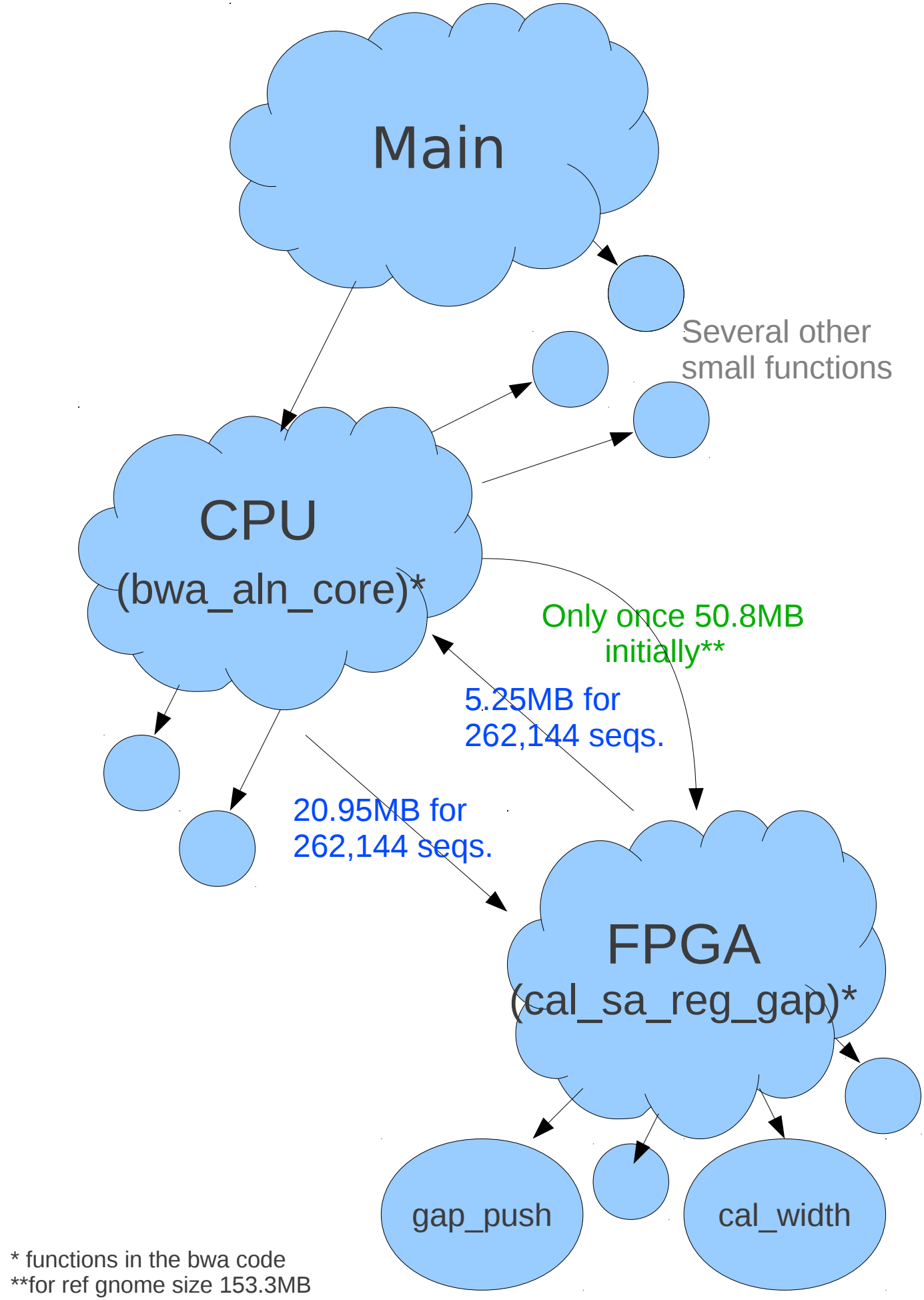

Figure1: Data flow graph of bwa-software tool

Steps	Process 1 (a)	Process 2(b)
1	CPU initially sends the reference gnome which is stored in DRAM	
2	CPU sends query sequences serially to DRAM	The reference gnome is serially transfered from DRAM to the FIFO buffers
3	Query sequences are transfered serially from DRAM to FIFO buffers	Step 2a continues
4	FPGA blocks(suppose to be iterative in CPU) begin executing on Ref gnome and their respective query sequence one-by-one. This output data is stored temporarily in DRAM	
5	Blocks get refreshed with new query sequences and the process continues	When CPU has finished sending the query sequences, it starts receiving the stored output data from DRAM

Table 6: Steps involved in the hardware-software codesign of bwa algorithm. The second process shown here is executed in parallel with the first. The steps are divided on the basis of time for execution.

Explanation of Figure 2:

1. CPU sends the reference gnome and the query sequence to the DRAM memory.
2. Reference gnome is transfered from DRAM to all the FIFO buffers simultaneously.
3. Ite blocks read the reference gnome data part by part from the FIFO Buffers.
4. Ite blocks read query sequence from DRAM*.
5. The Ite blocks process the data and send the result back to the DRAM memory**.
6. CPU receives the output data from DRAM.

*Due to less availability of I/O ports in DRAM, the data will be first written to BRAM
** Output Data will be first written to BRAM

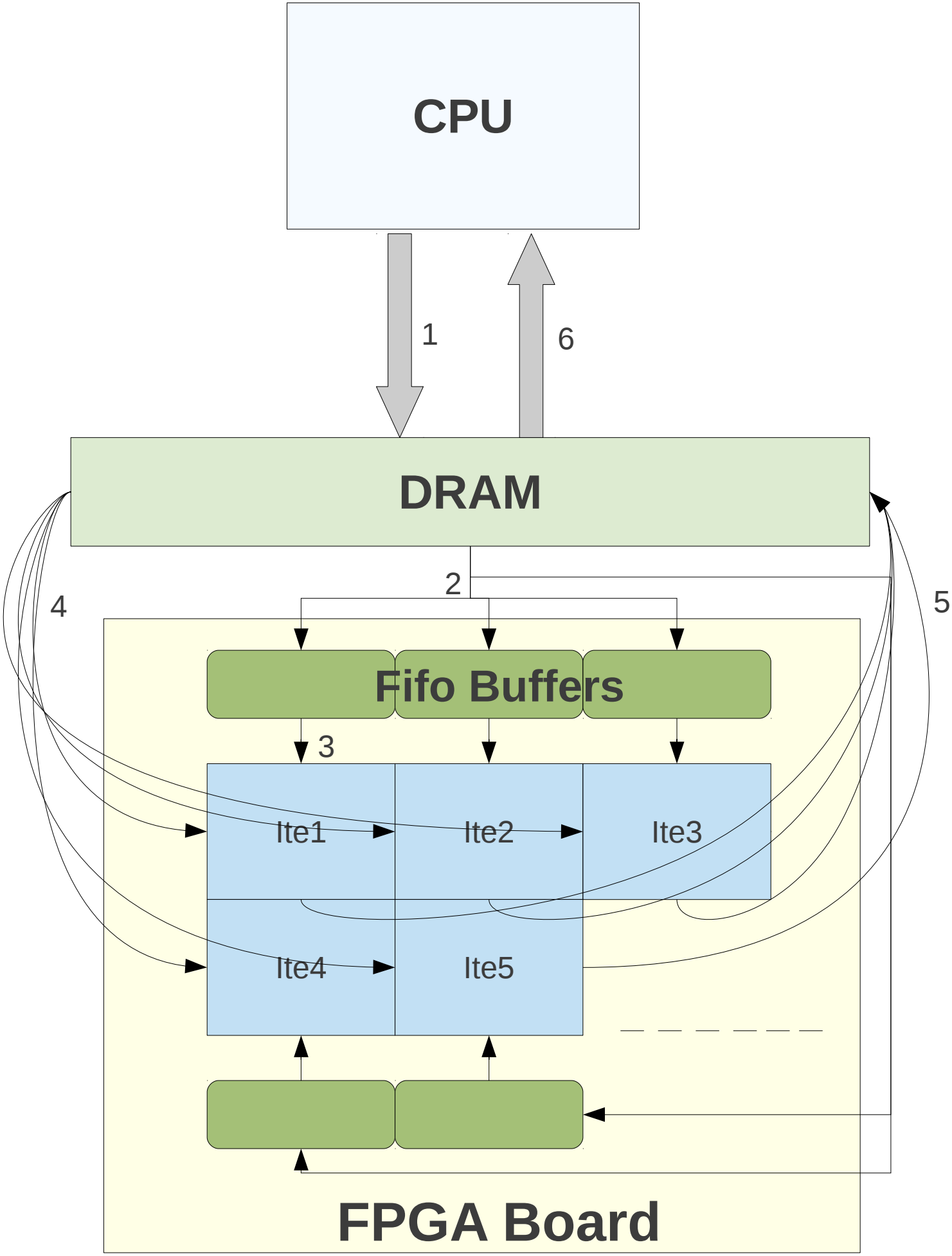

Figure2: Hardware-Software Codesign

Conclusion and Future Work

Sequence alignment is the first step to many applications in Bioinformatics and other related fields. Our work on FPGA may speed up the selected algorithm and thus fasten sequence alignment. This may result in speeding up of the entire chain of its applications in various fields such as biological research, diagnostic, biotechnology, forensic biology and biological systematics.

Our contribution may also help design better FPGA accelerators for next- generation sequencing. Our methodology may provide insight into hardware- software codesign of sequence alignment software tools. Furthur, our implementation can be compared with other possible hardware implementations like GPUs for a performance enchancement.

References:

[1]- Martinez, J.; Cumplido, R.; Feregrino, C.; Dept. of Comput. Sci., National Inst. of Astrophys., Opt. & Electron., Puebla, Mexico: An FPGA-based parallel sorting architecture for the Burrows Wheeler transform. Reconfig urable Computing and FPGAs, 2005. ReConFig 2005 30 Sept. 2005.

[2]- Heng Li and Nils Homer A survey of sequence alignment algorithms for next-generation sequencing Brief Bioinform 2010 11: 473-483.

[3]- FPGA implementation of MD5 hash algorithm, J Deepakumara

[4]- Svetlin Manavki and Giorgio Valle, CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment BMC Bioinformatics, vol. 9, no. Suppl 2, pp. S10, 2008.

[5]- Yupeng Chen, Bertil Schmidt, and Douglas L. Maskell. A Reconfigurable Bloom Filter Architecture for BLASTN, in ARCS, 2009, pp. 40-49.

[6]- Li, I. T., Shum, W., and Truong, K. 2007. 160-fold acceleration of the smith-waterman algorithm using a field programmable gate array (fpga).

[7]- Peiheng Zhang , Guangming Tan , Guang R. Gao, Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform, Proceedings of the 1st international workshop on High-performance reconfigurable computing technology and applications: held in conjunction with SC07, November 11-11, 2007, Reno, Nevada.

[8]- Li H, Durbin R. Fast and accurate long-read alignment with Burrows Wheeler transform. Bioinformatics 2010; 26(5):589-95.

[9]- Samtools [http://samtools.sourceforge.net/]

[10]- Intel® Vtune™ Amplifier XE Performance Profiler [http://software.intel.com/en us/articles/intel-vtune-amplifier-xe/]

[11]- UCSC Genome Bioinformatics [http://hgdownload.cse.ucsc.edu]

[12]- Burrows-Wheeler Aligner [http://bio-bwa.sourceforge.net/]

[13]- MAQ: Mapping and Assembling with Qualities [http://maq.sourceforge.net/]

[14]- SOAP: Short Oligonucleotide Analysis Package [http://soap.genomics.org.cn/]

[15]- Heng Li and Nils Homer, A survey of sequence alignment algorithms for next-generation sequencing (2010) 11(5): 473-483 first published online May 11, 2010 doi:10.1093/bib/bbq015
[16]- Bao, Jiang et. Al, Evaluation of next-generation sequencing software in mapping and assembly, 2011 Apr 28, Journal of Human Genetics.
[17]-DNA sequencing From Wikipedia, the free encyclopedia. [Online]. Available: http://en.wikipedia.org/wiki/DNA_sequencing
[18] Sequence alignment From Wikipedia, the free encyclopedia . [Online]. Available: http://en.wikipedia.org/wiki/Sequence_alignment
[19]Smith, Temple F.; and Waterman, Michael S. (1981). "Identification of Common Molecular Subsequences". Journal of Molecular Biology 147: 195-197.
[20] Li H, Durbin R. Fast and accurate long-read alignment with Burrows- Wheeler transform. Bioinformatics 2010; 26(5):589-95.
[21] Langmead B, Trapnell C, Pop M, et al. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. Genome Biol ogy 2009;10:R25.
[22] Velvet: algorithms for de novo short read alignment using de Bruijn graphs. D.R. Zerbino and E. Birney. Genome Research 18:821-829.
[23] Li R, Yu C, Li Y, et al. SOAP2: an improved ultrafast tool for short read alignment. Bioinformatics 2009; 25:1966-7.
[24] Li H, Ruan J, Durbin R. Mapping short DNA sequencing reads and calling variants using mapping quality scores. Genome Res 2008;18:1851-8.
[25] Burrows M, Wheeler DJ. A block-sorting lossless data compression algorithm. Technical Report 124, Digital Equipment Corporation. CA: Palo Alto, 1994.
[26] Blumer A, Blumer J, Haussler D, et al. The smallest automaton recognizing the subwords of a text. Theoretical Computer Science 1985;40:31-55.
[27]Kent,W.J. (2002) BLAT: the BLAST-like alignment tool. Genome Res., 12, 656-664.
[28] Ning,Z. et al. (2001) SSAHA: a fast search method for large DNA databases. Genome Res., 11, 1725-1729.