Heterogeneous Cloud Framework for Big Data Genome Sequencing

Chao Wang, Xi Li, Peng Chen, Aili Wang, Xuehai Zhou, and Hong Yu

Abstract—The next generation genome sequencing problem with short (long) reads is an emerging field in numerous scientific and big data research domains. However, data sizes and ease of access for scientific researchers are growing and most current methodologies rely on one acceleration approach and so cannot meet the requirements imposed by explosive data scales and complexities. In this paper, we propose a novel FPGA-based acceleration solution with MapReduce framework on multiple hardware accelerators. The combination of hardware acceleration and MapReduce execution flow could greatly accelerate the task of aligning short length reads to a known reference genome. To evaluate the performance and other metrics, we conducted a theoretical speedup analysis on a MapReduce programming platform, which demonstrates that our proposed architecture have efficient potential to improve the speedup for large scale genome sequencing applications. Also, as a practical study, we have built a hardware prototype on the real Xilinx FPGA chip. Significant metrics on speedup, sensitivity, mapping quality, error rate, and hardware cost are evaluated, respectively. Experimental results demonstrate that the proposed platform could efficiently accelerate the next generation sequencing problem with satisfactory accuracy and acceptable hardware cost.

Index Terms—Short reads, genome sequencing, mapping, reconfigurable hardware, FPGA

1 Introduction

Next-generation sequencing (NGS) problems have attracted many attentions of researchers in biological and medical computing domains. The current state-of-the-art NGS computing machines are dramatically lowering the cost and increasing the throughput of DNA sequencing. Due to the heterogeneous accelerating approaches, Moore's law has largely fell behind the rate of performance improvement during the past decades, with no end in the foreseeable future. What's more, with the pace as progress in semiconductor technology in widespread and unexpected application fields, sequencing technology is being increasingly dominant especially across widely in scientific and medical domains, in particular from basic biology to forensics, ecology, evolutionary studies, agriculture, drug discovery, and the growing fields of personalized medicine [1].

DNA sequencing determines the nucleotide sequence of short DNA fragments, which consist of a small number (10 through 1,000) of bases, called short reads. This can be done in a massively parallel manner, yielding much higher throughput than older sequencing technologies—on the order of tens of billions of bases per day from one machine. For comparison, the human genome is approximately 3 billion bases in length, which would take months or years to be processed on a single machine.

Taking account the diversity of genome sequencing applications, obviously there is no general workflow worked for the tremendous NGS applications. However, one general approach derives the short reads by randomly fragmenting many copies of the genome is already familiar in cutting-edge methodologies. In these cases, the key first step in the data analysis pipeline is the short read mapping problem, as is depicted in [1]: determining the location in the reference genome to which each read maps best. The problem is technically challenging for two reasons. First, speed is becoming significantly important simply due to the volume of data. For example, in human genetic studies, mapping a billion reads from one subject to the human reference genome is a tedious and repeated routine.

Second, the sensitivity of the algorithm, which represents the ability to map sequences that are not completely identical to the reference, is an important issue. These differences exist both because of technical errors in the sequencing machines and because of genetic mutation of the subject genome. The latter case is rare but can not be ignored because it may help identify possible genetic disease risks. The cases are distinguishable because the sequencer errors are generally random while the genetic differences are not. Hence many mapped reads that consistently exhibit a difference with respect to the reference drives the desire for more and more reads, since more data gives more accurate variant calling.

Short read mapping is widely supported by open source software tools such as Bowtie [2], BWA [3], and BFAST [4],

- *C. Wang and P. Chen are with the Department of Computer Science, University of Science and Technology of China, Hefei 230027, Anhui, China. E-mail: {saintwc, qwe123}@mail.ustc.edu.cn.*
- *X. Li and X. Zhou are with the Suzhou Institute of University of Science and Technology of China, Suzhou 215123, Jiangsu, China. E-mail: {llxx, xhzhou}@ustc.edu.cn.*
- *A. Wang is with the School of Software Engineering, University of Science and Technology of China, Suzhou 215123, Jiangsu, China. E-mail: wangal@ustc.edu.cn.*
- *H. Yu is with National Center for Plant Gene Research, Institute of Genetics and Developmental Biology, Chinese Academy of Sciences, Beijing, China. E-mail: hyu@genetics.ac.cn.*

Manuscript received 11 Apr. 2014; revised 30 June 2014; accepted 17 July 2014. Date of publication 11 Sept. 2014; date of current version 30 Jan. 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TCBB.2014.2351800

running on a cluster of processors. However, NGS improvements are moving the bottleneck of the genome sequencing workflow from the sequencing phase to the short read mapping software. In particular, using MapReduce framework to accelerate the short read mapping process by exploiting the parallelism of the task has been proven to be an efficient way as CloudBurst [5] has shown. However, CloudBurst will not start until the millions of short reads have been imported into the system. This would be time-consuming, and any updates to one of the short reads require the MapReduce tasks to be re-run which will dramatically degrade the response time.

Furthermore, along with the rapid development of genome technology, in the foreseeable future, the commercial genome sequencing service could be provided to the research community. The requirements for scientific researchers to access the genome sequencing services provided by supercomputers would be a driven force to the genome technology in the future. To tackle this problem, a high performance genome sequencing engine that can response in a reasonable response time is becoming highly recommended and favorable.

Up to now, it has been popular and necessary to address the computational requirements for NGS using efficient hardware field programmable gate array (FPGA)-based acceleration. For example, the recent work in [6], [7] stream a good perspective of the reference genome through a system doing exact matching of the short reads. However, the "It didn't fit" problem from TC-FPGA community predictions [8] illustrates that the helpless fact the FPGA based heterogeneous computing platform is preferable to deal with computing-intensive applications, while tremendous amount of data may cause inevitable failure if only the hardware based acceleration is employed in the FPGA devices. Instead, they should be processed in a more parallel way, such as MapReduce [9] framework. During the past few years, there has been some creditable literature like FPMR [10] that integrates high level MapReduce parallel framework into FPGA based hardware prototype. However, in spite of the state-of-the-art hardware implementation achieving satisfying speedups, to our best knowledge, few of them have considered FPGA based MapReduce programming framework really applicable to this typical large amount data-intensive computing domains.

To address the above problems, in this paper we present a heterogeneous cloud framework with MapReduce and multiple hardware execution engines on FPGA to accelerate the genome sequencing problems. The contribution of this paper is claimed as below:

1) We propose a novel architecture to accelerate read mapping processing thread when facing large amount of requests with FPGA based MapReduce.
2) A distributed MapReduce framework is presented to dispatch the task into multiple FPGA accelerators. The Map and Reduce process is based on RMAP sequencing algorithm. And inside each FPGA based accelerator, we implement BWA-SW algorithm kernel in hardware that is connected to a local microprocessor to speedup the local alignment process.
3) We conduct a theoretical analysis on the MapReduce framework, and for each FPGA chip, we construct a hardware prototype subsystem. The experimental results with hardware speedup, sensitivity, quality, error rate and hardware utilization is presented and analyzed.

The structure of this paper is organized as below. We present the motivation and the related state-of-the-art studies and in Section 2, then Section 3 illustrates the architecture and FPGA based MapReduce programming framework. We also detail the execution flow of short read mapping in Section 3. Thereafter, we explore the hot spots of the genome sequencing application in Section 4. A theoretical study for both timing complexity and speedup analysis is also described in Section 4. In Section 5, we present the hardware prototype and detail the experimental results on the real FPGA hardware platform. Finally, Section 6 concludes the paper and introduces some future works.

2 Related Work and Motivations

There are quite a lot of successful short read mapping software tools that address the problem of processing the enormous amount of data produced by the next-generation sequencing machines. In this section, we first present the motivation, and then analyze the state-of-the-art related work of short read mapping algorithm, MapReduce acceleration frameworks and heterogeneous accelerating approaches respectively.

2.1 Motivation

Although it has been only less than five years since the emergence of the short read mapping problem of genome sequencing, it is clearly an area of intense research activity. It is widely acknowledged this field to be one of the pioneering application domains that benefit from innovative ideas.

Due to the I/O and computing intensive requirements of state-of-the-art DNA sequencing problem, we anticipate that DNA sequencing programs will be located in supercomputers but in a manner that provides pervasive and ubiquitous services to scientific researchers or even public users. In this scenario, system performance assessment needs to consider more metrics such as response time, scalability and mismatching rate, rather than the raw parallel speedup oriented, as is illustrated in most of the published work. In a cloud service model, it is likely that the service need to begin operation before all the required data is completely transferred via internet.

To the best of our knowledge, this approach targeting NGS has not yet been fully studied. In this paper, we present a novel approach for solving NGS problems using a cloud based paradigm. We apply cluster computing approach to FPGA based multiple acceleration engines into a sound framework, and discuss architecture in a service-oriented manner. We also apply MapReduce programming model into the hybrid system to manipulate FPGA based accelerators as function units, which could largely facilitate the programmers to improve the performance.

2.2 Current Short Read Mapping Algorithms

In this section, we explore the state-of-the-art short read mapping solutions, in particular into two main algorithmic categories, as is inherited from [1]. The first category of

TABLE 1
Summary for Accelerations for Genome Sequencing

Type	References	Performance and Speedup	Algorithms or Methods	Features
GPU	[16]	2∼30x	Unspecified	Based on nVidia CUDA Model
	[17]	3.5∼10x	Smith-Waterman	MUMmerGPU SW with CUDA
	[18]	13x	Smith-Waterman	NVCC compiler With GPGPU
	[19]	17 (30) GCUPS	Smith-Waterman	SIMT and virtualized SIMD
FPGA	[6]	10x	Smith-Waterman	Parallel Mapping Single FPGA
	[7]	1.6∼4x	RMAP Algorithm	A simple Design Space Exploration
	[20]	1.86 vs Bowtie	Compact Linear Systolic	Up to 3,574 matchers in parallel
	[21]	22∼42.9x	PerM Algorithm	Up to 100 PEs integrated
	[22]	80∼190x	Based on BLAST SW	Apply prefiltering to BLAST prefiltering
	[1]	31vs Bowtie	Based on BFAST SW	Multiple Smith-Waterman Engines
	[23]	Up to 150x	Based on MUSCLE SW	MUDISC and MUSCLE introduced
	[24]	2.4x vs GASSST	Block-wise Alignment	Both SW and FPGA based HW
	[25]	185∼250x	Smith-Waterman	With up to 384-PEs integrated
MapReduce	[5]	30x vs RMAP in Serial	RMAP Algorithm	Hadoop Clusters with 24 Nodes

solution is based upon a block sorting data compression algorithm called the Burrows-Wheeler transform (BWT) [11]. This solution uses the FMindex [12] to efficiently store information required to traverse a suffix tree for a reference sequence. These solutions can quickly find a set of matching locations in a reference genome for short reads that match the reference genome with a very limited number of differences. However, the running time of this class of algorithm is exponential with respect to the allowed number of differences; therefore BWT-based algorithms tend to be less sensitive than other creditable solutions. Bowtie [2] and BWA [3] are examples programs based on this algorithmic approach.

The second category of solution leverages the fact that individual genomes differ only slightly, meaning it is likely that some shorter sub-sequences of a short read will exactly match the reference genome. This technique is called seed-and-extend and these shorter sub-sequences are called seeds. For example, for a 30 bp read to map to a reference with only one difference, there must regardless of where the difference occurs. In fact, if a we align an m bp read with at most k differences, there must exist one exact alignment of $m/(k+1)$ consecutive bases [13]. An index of the reference genome is compiled first, which maps every seed that occurs in the reference genome to the locations where they occur. To align a short read, all the seeds in the read are looked up in the index, which yields a set of candidate alignment locations (CALs). The short read is then scored against the reference at each of these CALs using the Smith-Waterman [14] string-matching algorithm. The location with the highest score is chosen as the alignment location for a short read. For example, BLAST [4] uses a hash table of all fixed length k-mers in the reference to find seeds, and a banded version of the Smith-Waterman algorithm to compute high scoring gapped alignments. RMAP [15] uses a hash table of non-overlapping k-mers of length $m/(k+1)$ in the reads to find seeds.

Based on the string match problem abstractions, plenty of traditional approaches those tackle high-speed similarity analysis and indexing exploration can be adopted, such as [26], [27], [28], [29], [30]. Baker and Prasanna [31] presented a hardware implementation of the Knuth-Morris-Pratt (KMP) algorithm. Since the KMP algorithm is designed to match the input stream against a single string, one matching unit is required per string, and the hardware system is composed of a linear array of matching units. The methods presented in [32] and [33] are based on pre-decoded characters with hardwired logic circuits. The system is optimized with respect to the given set of signatures and the characteristics of the FPGA devices. The FPGA is reconfigured when there are changes to the signature set. However, the long latency required in offline generation of the optimized hardwired circuits is considered as a major disadvantage in a network intrusion detection system that demands fast responses to hostile conditions. For the sake of FPGA accelerators, many of the proposed hardware solutions are based on the well-known Aho-Corasick (AC) algorithm [34], where the system is modeled as a deterministic finite automaton. The AC algorithm solves the string matching problem in time linearly proportional to the length of the input stream. However, the memory requirement is not feasible in a straightforward hardware implementation. In particular, [35] presents a pipelined processing approach to the implementation of AC algorithm. However, these approaches are not appropriate to be implemented in hardware due to the complexity and timing overheads.

2.3 Accelerating Approaches

Along with the novel short read mapping algorithms, multiple attempts have also been conducted to accelerate short read mapping in diverse high performance computing techniques. Graphics processing units (GPUs), MapReduce and FPGA based hardware are the most widely used acceleration engines. Table 1 listed most of the state-of-the-art literatures for type, reference, performance metric, utilized algorithm, and special features. In particular, the devoted researches can be divided into following categories:

2.3.1 GPU Based Accelerators

GPUs have recently been used as a mature approach for several bioinformatics applications, especially for sequence alignment, one of the most significant research areas. For example, [16] presents a similarity using Smith-Waterman algorithm in GPU accelerators, using compute unified device architecture (CUDA) programming engines.

MUMmerGPU [17] is an open-source parallel pairwise local sequence alignment program that runs on commodity GPUs in common workstations. Based on this research, [18] uses features a stackless depth-first-search print kernel with massive GPU data layout configurations to improve register footprint and conclude higher occupancy. Liu et al. [19] makes new contributions to Smith-Waterman protein database searches using compute unified device architecture. A parallel Smith-Waterman algorithm has been proposed to further optimize the performance based on the single-instruction-multiple-thread (SIMT) abstraction.

2.3.2 FPGA Based Accelerations

Nevertheless, numerous attempts to accelerate short read mapping on FPGAs tried to use a brute-force approach to compare short sequences in parallel to a reference genome. For example, the work in [6], [7] streams the reference genome through a system doing exact matching of the short reads. Knodel et al. [6] demonstrate a greater sensitivity to genetic variations in the short reads than Bowtie and MAQ, but the mapping speed was approximately the same as that of Bowtie. Also, this system demonstrated mapping short reads to only chromosome 1 of the human genome. Fernandez et al. [7] demonstrate between 1.6x and 4x speedup versus RMAP [15] for reads with between 0 and 3 differences. This implementation was for the full human genome.

Some recently papers conducing DNA short read sequencing problem using FPGA based reconfigurable computing acceleration engines are proposed in [1], [20], [21], [22], [23], [24], [25]. Of these approaches, [20] proposes a systolic custom computation on FPGA to implement the read mapping on a massively parallel architecture. This literature enables the implementation of thousands of parallel search engines on a single FPGA device. Wang et al. [36] shares the idea of genome sequencing using MapReduce on FPGA with multiple hardware accelerators, and further presents the Big data genome sequencing on Zynq based clusters [37]. Moreover, the authors in [21] proposed a CPU-FPGA heterogeneous architecture for accelerating a short reads mapping algorithm, which was built upon the concept of hash-index with several optimizations that reorder hash table accesses and compress empty hash buckets. The authors in [22] applies pre-filtering of the kind commonly used in BLAST to perform the initial all-pairs alignments. In [1] the authors proposed a scalable FPGA-based solution to the short read mapping problem in DNA sequencing, which greatly accelerates the task of aligning short length reads to a known reference genome. Not only the first stage of progressive alignment in multiple sequence alignment problem, the third stage of progressive alignment on reconfigurable hardware in [23]. Chen et al. [24] introduces a hybrid system for short read mapping utilizing both software and field programmable gate array-based hardware. Zhang et al. [25] presents a implementation of the Smith-Waterman algorithm for both DNA and protein sequences on the platform. The paper introduces a multistage processing element design and a pipelined control mechanism with uneven stage latencies to improve the performance and decrease of the on-chip SRAM usage. Recently, [38] uses FPGA-based system to accelerate the next generation long read mapping, but the hardware engines is implemented on a sole FPGA board. Chen et al. [39] presents a novel FPGA-based architecture which could address the problem with a bounded number of PEs to realize any lengths of systolic array. It is mainly based on the idea of the banded Smith-Waterman but with a key distinguish that it reuses the PEs which are beyond the boundary.

Previous efforts doing short read mapping using FPGAs have achieved at most an order of magnitude improvement compared to software tools. Also, previous solutions are not convincible to produce a system that is well-suited to large scale long read mapping and full genome sequencing. Finally, current literatures only uses one unique method on FPGA to accelerate the short read mapping problem, therefore it could make the most advantage of the parallel computing and hardware acceleration based techniques.

2.3.3 MapReduce Based Acceleration Frameworks

Besides the above heterogeneous accelerating engines, MapReduce [9] is an alternative software framework developed and used by Google to support parallel-distributed execution of data intensive applications. Utilizing MapReduce programming framework, CloudBurst [5] is a new creditable parallel read-mapping algorithm optimized for mapping next-generation sequence data to the human genome and other reference genomes, for use in a variety of biological analyses including SNP discovery, genotyping and personal genomics. It is modelled after the short read-mapping program RMAP [15], and reports either all alignments or the unambiguous best alignment for each read with any number of mismatches or differences. This level of sensitivity could be prohibitively time consuming, but CloudBurst uses the open-source Hadoop implementation of MapReduce to parallelize execution using multiple compute nodes, which means it can deal with larger amount of short reads simultaneously. Similarly, [40] is based on the pre-process of the reference genomes and iterative MapReduce jobs for aligning the continuous incoming reads.

To sum up, even the MapReduce framework could disseminate the read mapping to multiple computing machines, but the execution kernel itself is quite inefficient, so the throughput of the entire system is still worth pursuing.

3. Short Read Mapping on Heterogeneous Cloud Framework

As the MapReduce framework has been successfully integrated into FPGA based researches [10], in this section, we demonstrate the system architecture of a hybrid heterogeneous system utilizing both software and field programmable gate array-based hardware for real time short reads mapping service. We will model it as a specific-domain search problem that has been studied well for the past decades.

3.1 High Level Architecture Framework

After sequencing DNA, researchers often map the reads to a reference genome to find the locations where each read occurs. The read-mapping algorithm reports one or more alignments for each read within a scoring threshold, commonly expressed as the minimal acceptable significance of

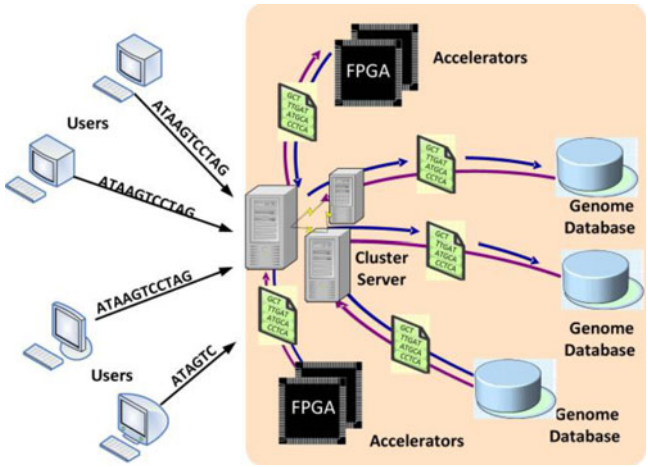


Fig. 1. Architecture framework for hybrid fystems.

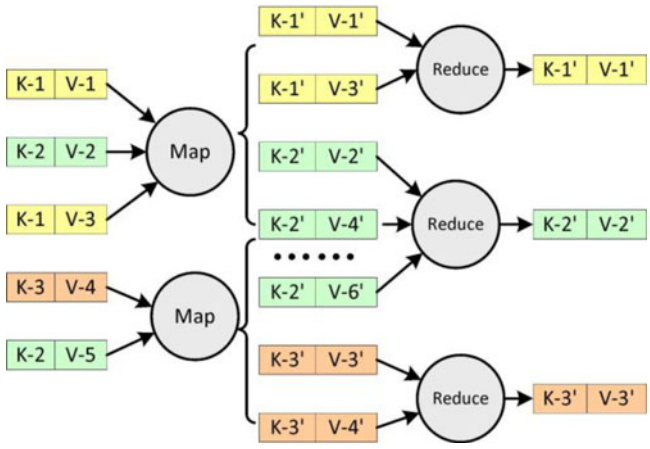


Fig. 2. Map and reduce phases in MapReduce framework.

the alignment, or the maximum acceptable number of differences between the read and the reference genome.

The motivation of combining the MapReduce and the FPGA is to utilize MapReduce framework for the big data aspect and the FPGA for the acceleration part. In particular, growing with the data amount of genome sequencing, it will be essential to use MapReduce to handle the extremely large amount of data. MapReduce framework, in many occasions, has been long proved and demon-strated as an efficient methodology. In this paper, to benefit from both MapReduce and FPGA, we construct the MapReduce framework on CPU/FPGA hybrid platforms. Normally the process is divided into following two steps:

1) *The first stage is the MapReduce stage*. Scientific researchers can send short reads as a stream into our system through the MapReduce server. As soon as the request is received, it will undergo a general Map scheduling stage, which partitions the entire genome sequencing task into many small jobs, and then distributes them to parallel computing nodes.
2) *The second stage is the local alignment in FPGA*. Multiple hardware acceleration engines are deployed to speedup the genome sequencing analysis procedure.

Our proposed architecture framework is illustrated in Fig. 1. Generally the system is constructed on a central cluster server which is responsible for sequence pre-processing, database access and user interaction, while multiple hardware acceleration engines are deployed to speedup the genome sequencing analysis procedure.

Throughout this paper, we use Hbase [41] as the Genome Database, which is a distributed storage system for random, real time read/write access to our Big Data for very large tables—with billions of rows and millions of columns. Hbase provides big table-like capabilities on top of Hadoop [42] and HDFS [43]. Data stored in Hbase was divided according to the table where it belongs. Each table may have billions of rows and all the rows in the table were ordered by the key field. The biggest difference between Hbase and other SQL database is that Hbase is column-orient. Due to the large scale at millions of columns, each column belongs to column-family, which is a logic union of columns. Each cell in Hbase stores multi versions of the data. Since one table can contains hundreds or thousands column families and columns in each column family can be millions. This property is very important for our system design. Each cell also contains multi version values that allow us remember the history information.

At start-up, the cluster server is ready to receive multiple user requests at any time. Due to the large amount short reads (millions at least) in general, it is not practical to start the mapping procedure in one machine, even in assistance of hardware acceleration engines. We also need to distribute the short reads to different computing machines in parallel, each of which is equipped with FPGA based accelerators. As a consequence, we extend the MapReduce programming framework with multiple hardware acceleration engines on single FPGA to further reduce the execution time for the big data genome sequencing processing applications.

3.2 MapReduce Distribution Model

MapReduce [9] is a well known software framework developed and used by Google to support parallel-distributed execution of their data intensive applications. Google uses this framework internally to execute thousands of MapReduce applications per day, processing petabytes of data, all on commodity hardware. Unlike other parallel computing frameworks, which require application developers explicitly manage inter-process communication, computation in MapReduce is divided into two major phases called *map* and *reduce*, and the framework automatically executes those functions in parallel over any number of processors. A demonstration work flow of MapReduce framework is illustrated in Fig. 2.

A typical MapReduce framework includes two phases: *map* and *reduce*. The *map* function computes key–value pairs from the input data based on any relationship applicable to the problem, including computing multiple pairs from a single input. Once the mappers are complete, *MapReduce* shuffles the pairs so all values with the same key are grouped together into a single list. The grouping of key–value pairs effectively creates a large distributed hash table indexed by the key, with a list of values for each key. The reduce function can be arbitrarily complex, but must be commutative, since the order of elements in the key-value list is unstable. As an optimization, *MapReduce* allows reduce-like functions called *combiners* to execute in-memory

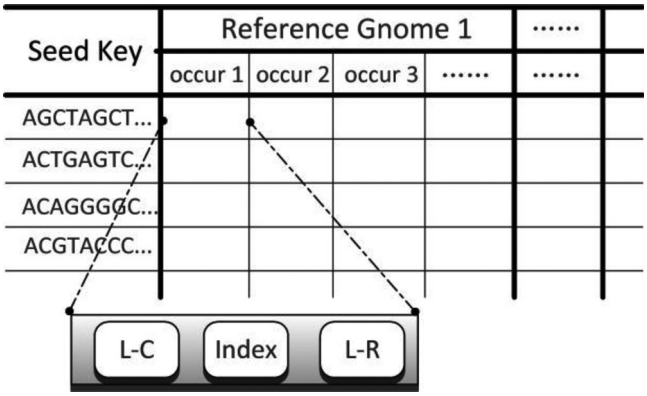


Fig. 3. Reference genome lookup table. Index means this occurrence's position in the reference genome 1, L-C means the left elements of this occur; L-R means the right elements of this occur; The size of L-C and L-R should be m-s+k (k is the maximum number of differences or mistakes; s is seed length, m is minimum length of the reads.

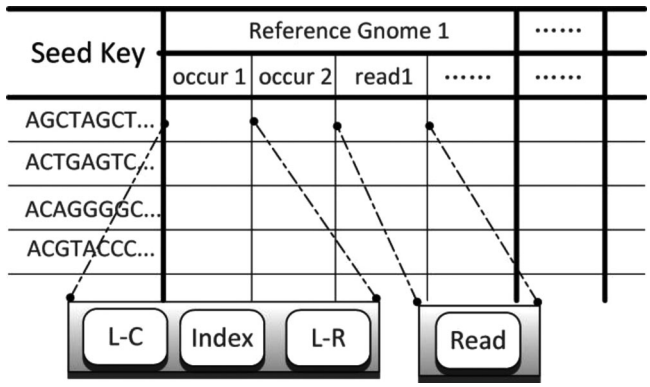


Fig. 4. Hbase table after users short read requests written.

immediately after the *map* function. *Combiners* are not possible in every application because they evaluate on a subset of the values for a given key, but when possible, reduce the amount of data processed in the shuffle and reduce phases. In the k-mer counting example, the *combiner* emits a partial sum from the subset of 1 s it evaluates, and the *reduce* function sums over the list of partial sums.

The MapReduce programming framework has been successfully introduced with respect to the FPGA based accelerations. For the sake of massive data processing requirement, we use RMAP algorithm to align the short reads. RMAP algorithm is based on seed-and-extend approach, which is composed of two phases: Compare Phase and Seed-Extend Phase [15]. Of the two phases, Compare Phase is basically considered as a mathematical string matching problem, which in this scenario is in charge of locating the seed string in the reference genome sequence, therefore it is regarded as the time-consuming part. On the other hand, Seed-Extend Phase will be processed after the seed are indexed, and the extensions in both sides of the seed string are to be compared to explore whether a mismatch (e.g. genome mutation) appears. Due to that the length of extended string is always much shorter than the reference itself, hence the timing complexity of Seed-Extend Phase can be ignored with respect to the large amount of the reference sequence. Moreover, for the sake of the area and hardware constraints of FPGA chip areas, it is not feasible to implement all the hardware logic within limited hardware look-up-tables (LUTs) and slices registers, consequently we put our concentrations on the hardware accelerations engines of the Comparing Phase, implementing hardware IP cores on FPGA for RMAP string matching applications.

To design the RMAP algorithm into MapReduce programming framework, both the map and shuffle stages are employed in string compare operations, while the reduce stage is in charge of seed-extend operations.

At the very beginning of our system, we use MapReduce tasks to build a *lookup table* for all the seed from the reference genomes like Fig. 3. Each seed may appear multiple times in one reference genome, so we make every reference genome as a column-family in Hbase table and each occurrence of a given key is a column in the family. The benefits to organize this way is we can make sure each reference genome information was stored in one physical server, which would accelerate computing a lot.

Considering the data structure of reference genome lookup table structure, we utilize the LUT based slice registers within FPGA development board to allocate block parameters, as it is the natural way to access the L-C, Index and L-R values in hardware acceleration. However, due to the hardware resources constraints, we can not integrate all the Lookup Table together, therefore they are implemented as private memory access for different hardware accelerators.

When a short read reaches the *buffer*, it would be written into the Hbase table immediately to form a table like Fig. 4 shows. It's the same table with Fig. 3 except the extra *read* columns that were added into the corresponding reference genome column-family. After writing into this table, each read should begin to align itself with the corresponding reference genomes. This processing is not complex but still time costing. To accelerate this procedure, we revise the Hbase *update* operations: whenever there is an update in specific column-family (which in our case is *read* column-family), there would be a new task allocated to do the aligning job. Due to localization of the same row and the same column-family, this task would not need communication, which will be faster with the plain MapReduce algorithm.

Beside the tasks executed when request arrives, we have a threshold on the *short read* buffer to judge whether the system load (means the short reads requests per minutes) was too high. In the situation that system load is high, each read will not be processed by individual task, instead, after writing the request reads into the Hbase table, we will schedule an global MapReduce task and generate the align results at the same time.

The scheduling and generation process is divided into three stages:

1) *Generate lookup table.* At the very beginning of our system, we got numerous reference genomes and we will schedule MapReduce tasks to generate the *seed lookup table*, which will be used to handle user requests. Fig. 5 shows an example of how the map, shuffle, and reduce function works in generating lookup table.

The map function emits k-mers of length s as seeds from all the reference sequences. The shuffle phase collects k-mers shared between the reference sequences. Finally, the reduce function write the same seed into the Hbase table.

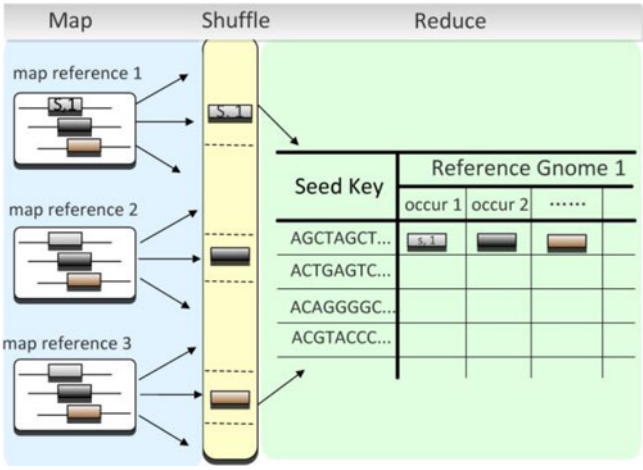


Fig. 5. Example on how to construct the lookup table.

2) *Individual update tasks*. The individual updates task is implemented with Hbase but modified in a data trigger way. Whenever updates on specify column-family happen (in our case, the *reads* family in *lookup table*), a new process will be generated on the node where the reference genome stores. As the read has been stored in the reference genome column-family, the process can read the request reads locally.

After obtaining the request reads and the left flank and right flank of the seed, we can extend the exact alignment seed into longer inexact alignments. We partition the reference genome information into the set R and partition the read into set Q. Then attempt to extend each pair of tuples from the Cartesian product R *X* Q using either a scan of the flanking bases to count mismatches, or the Landau-Vishkin k-difference algorithm for gapped alignments. If an end-to-end alignment with at most k mismatches or k difference is found, it is then checked to determine if it is a duplicate alignment.

3) *Global MapReduce tasks*. The global MapReduce task will be scheduled when the system suffers from high volumes of requests and when most of the requests do not hit the cache. The global MapReduce job is scheduled on the *lookup table*, where all the reference genomes and the reads were stored. The Map-Shuffle-Reduce phases are pretty similar with the way we build the *lookup table* except that we need to consider the reads stored in each reference column-family. In the Reduce phase of the global MapReduce task, we use the same algorithm as the individual update task does.

The above section presents how MapReduce framework is deployed to handle the short read processing problem. In our solution, multiple acceleration engines have been integrated to achieve higher speedup. As both the map and shuffle stages are employed in string compare operations, while the reduce stage is in charge of seed-extend operations.

3.3 Local Alignment on Single FPGA Chip

The basic unit of the MapReduce framework is one single FPGA chip. In each chip, the accelerator is attached to a local microprocessor. In order to support fast data transfer, every function unit has been packaged into a same manner accessed by Xilinx FSL channels. For demonstration, state-of-the-art Xilinx early access partial reconfiguration paradigm is introduced to support IP core replacement.

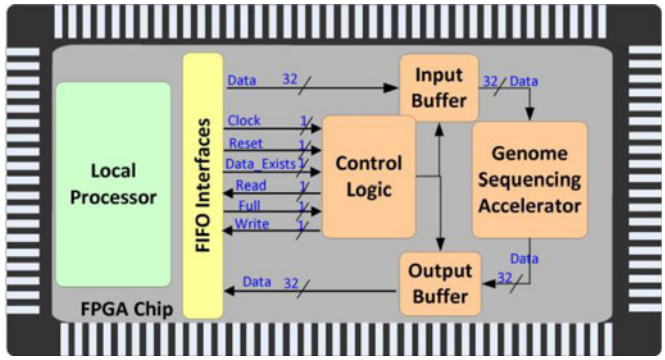


Fig. 6. Architecture in single FPGA chip, the cmmunication interfaces between processor and accelerator is based on FIFO interface.

The reconfigurable fabric communicates with the main core through a set of first-in-first-out (FIFO) interfaces as shown in Fig. 6. The FIFO interfaces are connected to/from the commit stage of the main core pipeline. The core-to-accelerator interface works to enable communication between the core and reconfigurable fabric. The processing core sends its execution trace to the genome sequencing accelerator via the FIFO interface, so that the fabric can perform the operations on each forwarded task.

A forward FIFO sends a trace of instructions, which are completed and ready to commit, in the program order. A FIFO packet contains fairly comprehensive information, including the Data, Clock, Reset, Exist Full, Write signals. The FIFO packet also includes the input buffer, output buffer and control logic, in order to manipulate the finite state machine.

The accelerator needs to be able to raise an interrupt and communicate with the main core through explicit instructions. The instructions may read/write configuration registers on the co-processor and/or perform custom operations for each extension.

The main core and hardware sequencing accelerator communicate with each other through a set of FIFO interfaces. The core-to-accelerator interface uses a Xilinx Fast Simplex Link as a demonstration. Signals in both directions include Control and FIFO data signals. There are signals such as Clock and Reset shared by both main core and IP core.

Communication between main core and accelerator is utilized in master-slave manner. For task distribution, microprocessor acts as master which sends a specific instruction to drive the FIFO based interconnection.

Accelerator acts as masters when a back FIFO (BFIFO) sends a return value to microprocessors. Note that the proposed FIFO interface with the reconfigurable fabric can easily support IP core reconfiguration on the main core for each extension. For example, after the hardware implementation of the accelerator is replaced, the communication interfaces remain the same. The unified interfaces allow applications to substitute accelerators during task execution for different tasks, without interrupt the task execution. A reconfiguration controller is introduced to deal with the hardware reconfiguration.

TABLE 2
Parameters Used to Analyze Speedup

Symbols	Description	Typical
m	Minimum short reads size	100 ~ 2,000 bp
k	Maximum mismatches	4 (<10)
s	Seed size from reads	m /(k + 1)
n	Reference genome size	1 billion
R	Number of Reads	1 million
p	Speedup with multiple HW	75 × c

3.4 Sequencing Kernel As Hardware Accelerator

The aim of the sequence alignment is to report the matched locations of queries (reads are usually called the queries when conducting the alignment) in the reference. In order to accelerate the read mapping problem, we employ a local sequencing kernel in each hardware accelerator. This specific task is implemented as a hardware kernel for genome sequencing accelerator in the FPGA chip. For demonstration, we employ BWA-SW [44] as the kernel sequencing accelerator. Please note that the kernel could be replaced by other state-of-the-art genome sequencing engines. Here, the alignment procedure of BWA-SW algorithm contains three major sequential steps: build index, search seeds and local alignment, respectively. As the algorithm should be familiar with the readers, consequently we only illustrate the brief introduction of each step of the sequencing process:

1) *Build index*. To scan the enormous long reference efficiently, indexes are constructed for the queries and the reference. The index for the reference is only built once and can be shared among all queries.
2) *Search seeds*. After then, seeds are searched by traversing the indexes of the queries and reference with dynamic programming. Here, seeds mean those small identical substrings between the queries and the reference which indicate more possibility to be in the exact alignments.
3) *Local alignment*. The seeds are extended and compared with the local alignment algorithm. A similarity score is calculated by the local alignment algorithm to clarify whether the places are the real alignment locations or not. Finally, the information of the mapped reads is recorded in the sequence alignment file.

4 Theoretical Analysis

In this section, we analyze the timing complexity of the algorithm, and then present the ideal speedup using different hardware architectures.

4.1 Timing Complexity Analysis

The terms and parameters used in the equations are defined in Table 2. m refers to the minimum short read size, which lies between 100 to 2,000 bp generally. k is the maximum allowed mismatches, which is configured to 4. s and n represents the seed size and reference genome size, respectively. Due to the large amount of human genomes, the reference genome size could reach to 1 billon or more. R indicates the number of short reads, which could also be up to 1 million.

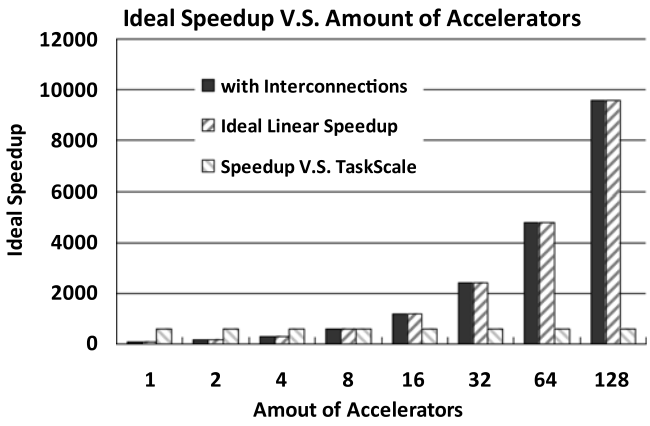


Fig. 7. Theoretical speedup versus amount of accelerators.

Finally, the speedup with multiple hardware accelerators is set to 75 × c, while c is the number of accelerators.

Assume m refers to the minimum short reads size, k stands for the maximum allowed mismatches, s indicates the seed size, n denotes the reference genome size, and R represents the number of short reads. Based on these parameters and the processing flow of the algorithm, the timing complexity is analyzed separately in the following two phases:

1) *Compare phase.* In serial algorithm, each read was divided into m/s seeds, and each seed compares with the reference genome to get index, the complexity would be $O(n + s)$, therefore the timing complexity in this stage is $O(mn + m)$.
2) *Seed and extend phase.* After that, we uses a variation of the Landau-Vishkin k-difference alignment algorithm which cost $O(km)$ to get the score of each seed. Considering the m/s seeds, the timing complexity in this stage is $O(km^2)$.

So the total time would be $O(m^*n + km^2 + m)$. For R reads, the total runtime would be $O(Rmn + Rkm^2 + Rm)$. It is widely applied that the reference genome size n is much bigger than the minimum short read size m, consequently the final timing complexity is $O(Rmn)$.

Based on the timing analysis, we can derive the speedup as we use the both the hardware accelerators and MapReduce parallel processing framework. Let c be the nodes number of the acceleration engines, and SPU refers to the speedup string match hardware implementations, hence we can achieve the speed by multiple hardware acceleration engines under MapReduce framework $p = SPU \times c$. Finally the total time of R reads in our system should be less than $O(Rmn/p)$.

4.2 Speedup Analysis

We have evaluated the curve depicting how the speedup of the hybrid system is affected by the number of acceleration engines. Due to the large amount of the genome size n, the final speedup curve is quite linear with the slope coefficient is SPU (in our case, SPU is configured to 75x against software execution, according to the profiling results on preliminary experiments), as is presented in Fig. 7.

Both curve for binary tree and ring topologies appear to be extremely close to the ideal linear speedup. The reason is

that as the topologies take $log\ c$ and $c/2$ for translation delays. When c is relatively small than the data sets (n and m are quite huge due to human genome data sets), either approach does make any noticeable difference. The curve for both Binary Trees and Ring topologies are essentially coincident with the ideal linear speedup, therefore the theoretical speedup for 64 accelerators can reach to up to 4,800x at most. Fig. 7 also illustrates that the number of accelerators has a linear effect on the speedup when the genome size is much bigger than the short read size. We also measure the curve between speedup and the Task scales n. In this case, c is set to 8, and the speedup is fixed to $75^*8 = 600$, which is presented in the while bar in Fig. 7. The results demonstrate that the final speedup is irrelevant to the communication overheads as the application is computing intensive. Due to the liner speedup metrics, it is a favourable solution to combine both the high level parallel distributed with heterogeneous hardware acceleration engines into a sound framework for future biological DNA sequencing solutions.

5 Experimental Results and Analysis

5.1 Identify the Hardware Acceleration on FPGA

With respect to BWA-SW algorithm implemented on FPGA, it contains three major steps: constructing index, searching seeds and the local alignment. In order to save the area of the FPGA chip, we need to select which part is the bottleneck and should be accelerated. We first profile the algorithm on BWA-SW software to get an overview of the time consumption of these steps.

In the profiling step approximately 10,000,000 bp reads are generated in total by simulators with different lengths and error rates from the human genome database. In practical, we generate reads sets with the length of 100, 200, 500, 1,000, and 2,000 bp, respectively. For each kind of reads with different lengths, genome variations and gaps are randomly inserted with the error rate of 2, 5, and 10 percent, respectively. The total base pair amounts of all the test cases are almost the same (genome gaps may cause little difference for different cases). That is, there are 100,000 queries for reads of 100 bp, 50,000 queries for reads of 200 bp, 20,000 queries for reads of 500 bp, 10,000 queries for reads of 1,000 bp and 5,000 queries for reads of 2,000 bp. These generated reads sets are aligned back to the human genomes with BWA-SW software finally.

The profiling results are shown in Fig. 8 which outlines the time consumption features of each step in BWA-SW program. The x-axis indicates reads cases with different lengths. On the other hand, the y-axis describes the time percentage of Local Alignment step in the algorithm. In the figure s indicates the suffix array (SA) interval threshold. It clearly states that the local alignment phase takes the most time of the program. For these cases, the local alignment phase becomes the bottleneck of the program. In particular, we can get following two conclusions:

First, when the interval the interval threshold is set, the local alignment will contribute higher percentage of execution time with the short read lengths. For example, when $s = 7$, the execution time percentage of the local alignment phase takes 30.86 percent (at 100 bp), 57.09 percent (at 200 bp), 77.48 percent (at 500 bp), 77.7 percent (at 1,000 bp), and 71.09 percent (at 2,000 bp), respectively. The experimental results demonstrate that the timing complexity of the local alignment increases with the data scale sensitively.

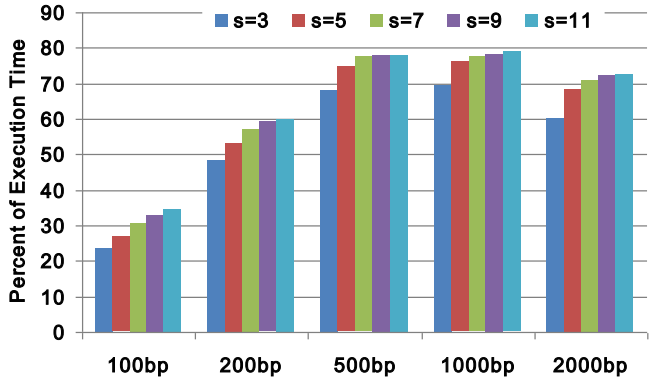


Fig. 8. Profiling results for aligning different reads sets to the human genome database. The figure reveals the time percentage of the Local Alignment step in BWA-SW program for different lengths of reads with different error rates. The profiling is conducted on a computer with the 64 bit Fedora13 OS installed and 2 Intel Xeon 2.53 GHz W3505 CPUs as well as a 6 GB main memory integrated.

Second, when the read length is set, the local alignment will also contribute higher percentage of execution time with higher interval thresholds. Similarly, when the read length is set to 1,000 bp, the execution time percentage of the local alignment phase takes 69.59 percent (at $s = 3$), 76.13 percent (at $s = 5$), 77.7 percent (at $s = 7$), 78.3 percent (at $s = 9$) and 79.06 percent (at $s = 11$), respectively. The experimental results demonstrate that the timing complexity of the local alignment increases with the interval threshold as well.

From the profiling results, we can get a conclusion that the local alignment phase is of the most desire to be accelerated and parallelized.

5.2 General Speedup on FPGA Prototype

To evaluate the parallel speedup of the MapReduce and FPGA architecture, we set up the real experimental environment on 8 Xilinx XV5 FPGA boards, each of which is equipped with one dual-core Microblaze processor, and hardware acceleration engines with programmable logic. On each microprocessor, a Linux operation system (version hanshuebner/linux-xlnx) with Hadoop framework is running smoothly. Meanwhile the one BWA-SW genome sequencing engine is implemented at register-transfer-level, which is encapsulated into hardware acceleration engines in each FPGA. The Microblaze processor is connected to the hardware through onchip FSL bus links. All the FPGA development boards are connected via Ethernets.

As all the FPGA development boards are deployed in the same configurations, therefore we only evaluate the speedup, sensitivity and other metrics on one FPGA chip subsystem. The improvement of the efficiency and the throughput of one FPGA chip can be revealed by the whole system speedup which is shown in Fig. 9. Approximately 10,000,000 bp reads are generated from the human genome database and aligned back with BWA-SW (version 0.6.2) and our acceleration platform.

The three polygonal lines in Fig. 9 show the speedup archived by our accelerating system over BWA-SW software for different lengths of reads with error rates of 2, 5,

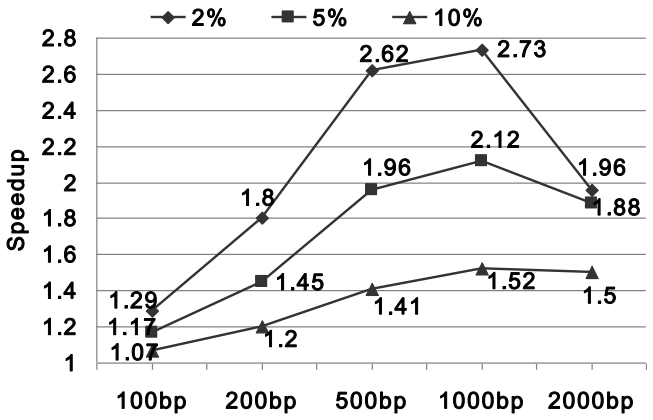


Fig. 9. Speedup of our system compared to BWA-SW software. The software runs on a computer with the 64 bit Fedora13 OS installed and two Intel Xeon 2.53 GHz W3505 CPUs as well as a 6 GB main memory integrated. Default options of the program are adopted for all the data sets. Reads sets with lengths shorter than 500 bp (include the 500 bp) are performed on the 500 bp configuration of our platform. Reads sets with lengths of 1,000 and 2,000 bp are carried on the 1,000 and 2,000 bp configurations of our platform, respectively.

and 10 percent, respectively. We can see the performance improvement of our platform for the short reads is quite little. The speedups range only from 1.07x to 1.29x for reads with lengths of 100 bp. And speedups for reads with lengths of 200 bp range from 1.2x to 1.8x.

However, for long reads, the speedups are higher than 1.4x and reach as high as 2.73x for the case of reads with lengths of 1,000 bp and 2 percent error rate. The speedups for the reads with lengths of 2,000 bp become a little lower compared with the ones of 1,000 bp for error rate of 2 and 5 percent, but the acceleration platform is still more than two times faster than BWA-SW software for these cases. The decrease of the speedups is caused by the increase of the apportion time of the fixed operations (some fixed operations in the program cost quite a lot of time while the number of queries for the 2,000 bp reads is half of that for the 1,000 bp). A trade-off exists between the single read alignment time cost and the total quantities.

5.3 Sensitivity Analysis

We improve the efficiency of the long read mapping by realizing the bottleneck of BWA-SW software via the hardware acceleration. It can be aware that this solution does not archive any advance on the sensitivity of the aligner. Nevertheless, the software itself has some parameters that affect the accuracy of the alignment. Users can improve the sensitivity of the aligner by finely tuning the options of the program. Generally speaking, the sensitivity can be increased at the plenty of the speed by loosening the seeds searching conditions. Fig. 10 illustrates the effects of the SA interval threshold and the z-best strategy option to our accelerating system. The command line used for alignment is as follows:

- bwa *bwasw -s s -z z human-index queries.fa*

where 'bwa' is the executable file. Option '*bwasw*' indicates the long reads mode is used. Option '-s *s*' reveals that the SA interval threshold is set to s ($s = 1, 3, 5, 7, 9, 11$ in Fig. 10). Option '-z *z*' means that the only the z number of the top best nodes of the inter loop of the nested traversal are concerned, denoted as $z = 1, 2, 3$ in Figs. 10, 10a, 10b and 10c respectively. Option '*human-index*' reveals the index files of the reference and option '*queries.fa*' is the filename of the reads set. It can be aware that bigger SA interval threshold s results in more valid seeds and thus increases the sensitivity of the program. On the other side, the bigger the z-best strategy threshold z, the more branches will be searched and therefore the more sensitive the program becomes.

From Fig. 10a, it can be obviously learnt that with the increase of the SA interval threshold s, the speedup archived by our accelerating platform also grows. Moreover, the tread becomes slower when the SA interval threshold gets bigger. However, for situations when the z-best

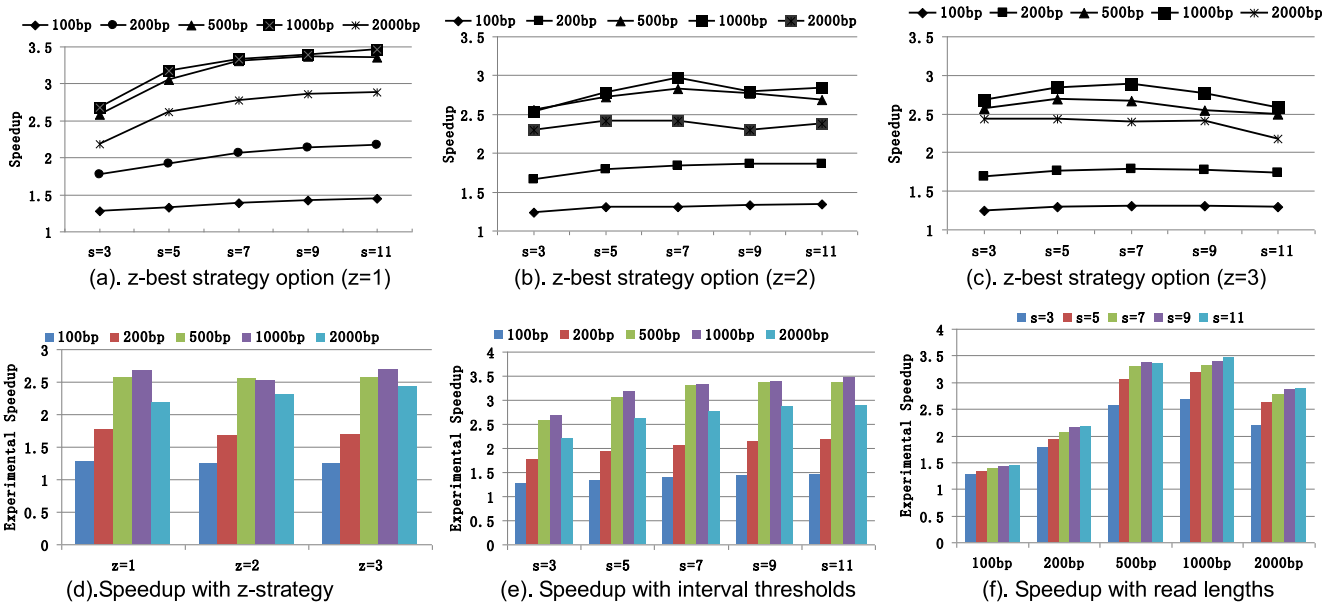


Fig. 10. Speedup of different data sets with different SA interval threshold s and z-best strategy threshold z. approximately 10,000,000 bp reads with lengths of 100, 200, 500, 1,000, and 2,000 bp are simulated. Genomes mismatches and gaps at the error rate of 2 percent are inserted in all these data sets. Reads sets in subfigure (a), (b), and (c) are the same.

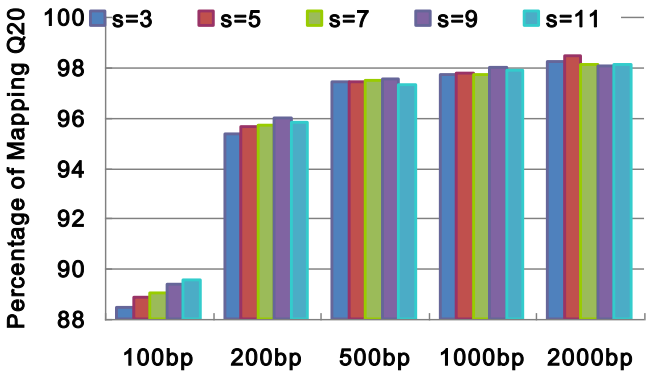


Fig. 11. Percentage of mapping quality scores >20.

strategy z equals 2 or 3, the change of the speedup along the SA interval threshold s becomes intangible. From Figs. 10b and 10c, it can hardly give a direction whether the performance will become better or worse when turning up the SA interval threshold.

The coordinate systems for all subfigures in Fig. 10 are the same. By taking account all the three subfigures in Fig. 10, a trend can be realized that the smaller the z-best strategy is, the higher the speedups are archived by the accelerating platform. Every polygonal line in Fig. 10a is a bit higher than the respective ones in Fig. 11b and the polygonal lines in Fig. 10b are also not worse than those in Fig. 10c. But not all test cases follow this trend. For example, the speedup in Fig. 10b for the case of 2,000 bp reads when the SA interval s equals 9 is 2.3x while the one in Fig. 10c comes a little bigger, to be 2.4x.

Moreover, Figs. 10d, 10e, and 10f illustrates the speedup with different z-strategy, interval thresholds, and read lengths, respectively. It can de derived that the speedup is not that sensitive with z-strategy, and increase much more with the interval thresholds and read lengths.

5.4 Quality and Error Analysis

Mapping quality [45] is a key concept of to estimate the probability of a query sequence being placed at a wrong position. In particular, the mapping quality scores quantify the probability that a read is misplaced. If an alignment algorithm guarantees to find all local alignments, mapping quality is determined by these local alignments only. However, as BWA-SW deploys heuristic rules, the chance of producing a wrong alignment is also related to the heuristics.

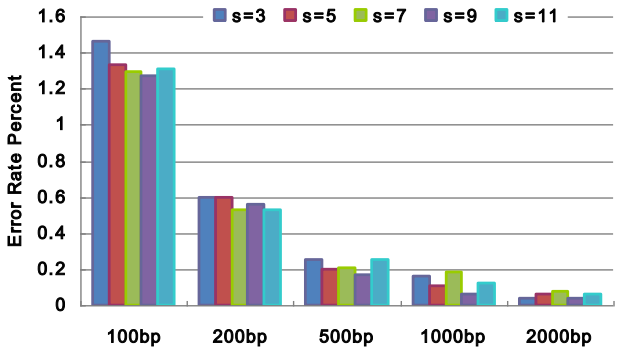


Fig. 12. Error rate different read lengths and error rates are simulated from the human genome.

TABLE 3
Hardware Utilization on FPGA (500 bp)

Resource	Used	Available	Percent
Slice Flip-Flops	55	1,35,168	0.04%
4 input LUTS	293	1,35,168	0.22%
Slice Logics	153	67,584	0.23%
Bonded IOBs	67	768	8.72%
BUFGs	1	32	3.13%

To estimate the mapping quality of a BWA-SW alignment, we measured the mapping quality scores with different read lengths and intervals.

Fig. 11 presents the percentage of the mapping qualities scores. We select Q20 as the major evaluation metric, as in the reference [45]. From the figure we can include following conclusions:

1) The value of Q20 grows with the growing read length when at a specific interval threshold, e.g. from 88.86 to 98.44 percent when s = 5;
2) The value of Q20 slightly increases with the interval threshold, e.g. the Q20 increase from 95.36 to 95.83 percent, when the read length is 1,000 bp.

Furthermore, in the implementation, we try to automatically adjust parameters based on the read lengths and sequencing error rates to make the configurations work well for inputs of different characteristics. Therefore we also need to explore the error rate for different read lengths and interval thresholds.

Fig. 12 depicts the percentage of the error rate. From the figure we can include that the error rates falls with the growing read length when at a specific interval threshold, e.g. from 1.33 to 0.06 percent when s = 5. Furthermore, when the read length is up to 2,000 bp, the error rates at all interval thresholds at insignificant level, ranging from 0.04 to 0.08 percent, respectively.

5.5 Hardware Cost

Due to the limitation of the chip area in FPGA, hardware cost is always a concerning factor for the FPGA based design paradigm. The chip area is always an issue of concern for the FPGA based design.

The prototype platform is implemented on the Xilinx Virtex-5 LX110T FPGA. Table 3 lists the hardware utilization on FPGA with the 500 bp case. As reported by the Xilinx tools, each banded Smith-Waterman module could operate at 200 MHz and costs 55 slice Flip-Flops, 293 4-input slice LUTs, and 153 for the slice logic. Bonded IO blocks are also needed for 67. With respect to the total hardware logic available on the FPGA chip, the percentage is quite acceptable that more than 100 accelerator modules can be integrated in one chip.

TABLE 4
Power and Thermal Information (2,048 bp)

Supply Power	Quiescent Power	Dynamic Power	Total Power
	1.319 W	0.019 W	1.339 W
Thermal Properties	**Max Ambient(°C)**		**Junction Temp(°C)**
	77.1		57.9

Furthermore, we also measured the power and thermal information in Table 4. The total power of a single accelerator is only 1.339 W, which includes a quiescent power at 1.319 W (98.6 percent), and a dynamic power at 0.019 W (1.4 percent). Meanwhile the max ambient and the junction temperature are 77.1 and 57.9 °C, respectively.

6 Conclusions and Future Work

This paper has proposed a general heterogeneous cloud framework for next genome sequencing with multiple hardware accelerators on FPGAs. We utilize an extended MapReduce distribution framework with multiple hardware accelerators on FPGA. By extending a distributed processing technique with hardware accelerators, this approach could bring significant speedup for genome sequencing alignment process. We have presented results both from a theoretical analysis and real hardware platform on Xilinx FPGA development board. The experimental results reveal that each hardware accelerator could achieve up to 2.73x speedup as well as it only occupies less than 1 percent of the FPGA chip resources.

Given the promising preliminary results illustrated in this paper, there exist various directions for future developments. Additional engineering effort needs to be applied to assess the scalability comparison to aforementioned cutting-edge approaches. Also a comparative study between the short read mapping applications in high performance GPU based supercomputers based on Hadoop clusters and detailed analysis of the feasibility, cost and overheads is working in progress. Furthermore, from the technical perspective, if different reads contain the same seeds, they could be cached in a fast table to response with high efficiency.

Acknowledgments

This work was supported by the National Science Foundation of China under grants (No. 61379040, No. 61272131, No. 61202053, No. 61222204, No. 61221062), Jiangsu Provincial Natural Science Foundation (No. SBK201240198), and the Strategic Priority Research Program of CAS (No. XDA06010403). The authors deeply appreciate many reviewers for their insightful comments and suggestions. Professor Xi Li is the corresponding author.

References

[1] C. B. Olson, M. Kim, C. Clauson, B. Kogon, C. Ebeling, S. Hauck, and W. L. Ruzzo, "Hardware acceleration of short read mapping," in *Proc. IEEE 20th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2012, pp. 161–168.

[2] B. Langmead, C. Trapnell, M. Pop, and S. L. Salzberg, "Ultrafast and memory-efficient alignment of short DNA sequences to the human genome," *Genome Biol.*, vol. 10, no. 3, p. R25, 2009.

[3] H. Li and R. Durbin, "Fast and accurate short read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 25, no. 14, pp. 1754–1760, 2009.

[4] N. Homer, B. Merriman, and S. F. Nelson, "BFAST: An alignment tool for large scale genome resequencing," *PLoS ONE*, vol. 4, no. 11, p. e7767, 2009.

[5] M. C. Schatz, "CloudBurst: Highly sensitive read mapping with MapReduce," *Bioinformatics*, vol. 25, no. 11, pp. 1363–1369, 2009.

[6] O. Knodel, T. B. Preusser, and R. G. Spallek, "Next-generation massively parallel short-read mapping on FPGAs," in *Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit. Processors*, 2011, pp. 195–201.

[7] E. Fernandez, W. Najjar, E. Harris, and S. Lonardi, "Exploration of short reads genome mapping in hardware," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2010, pp. 360–363.

[8] FCCM 2011 Top 10 Predictions for FCCMs in 2016. (2011). [Online] Available: http://www.fccm.org/2012/Previous.html

[9] D. Jeffrey and G. Sanjay, "MapReduce: Simplified data processing on large clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, 2008.

[10] Y. Shan, B. Wang, J. Yan, Y. Wang, N. Xu, and H. Yang, "FPMR: MapReduce framework on FPGA," in *Proc. 18th Annu. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, Monterey, CA, USA, , 2010, pp. 93–102.

[11] M. Burrows and D. J. Wheeler, *A Block-Sorting Lossless Data Compression Algorithm*. Palo Alto, CA, USA: Digital Equipment Corporation, 1994, p. 124.

[12] P. Ferragina and G. Manzini, "Opportunistic data structures with applications," in *Proc. 41st Annu. Symp. Found. Comput. Sci.*, 2000, pp. 390–398.

[13] R. A. Baeza-Yates and C. H. Perleberg, "Fast and practical approximate string matching," in *Proc. 3rd Annu. Symp. Combinatorial Pattern Matching*. 1992.

[14] T. F. Smith and M. S. Waterman, "Identification of common molecular subsequences," *J. Mol. Biol.*, vol. 147, no. 1, pp. 195–197, 1981.

[15] A. D. Smith, Z. Xuan, and M. Q. Zhang, "Using quality scores and longer reads improves accuracy of Solexa read mapping," *BMC Bioinformat.*, vol. 9, no. 128, pp. 1471–2105, 2008.

[16] S. A Manavski and G. Valle, "A compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC Bioinformat.*, vol. 9, no. Suppl. 2, p. S10, 2008.

[17] M. C Schatz, C. Trapnell, A. L Delcher, and A. Varshney, "High-throughput sequence alignment using graphics processing units," *BMC Bioinformat.*, vol. 8, p. 474, 2007.

[18] C. Trapnell and M. C. Schatz, "Optimizing data intensive GPGPU computations for DNA sequence alignment," *Parallel Comput.*, vol. 35, nos. 8/9, pp. 429–440, 2009.

[19] Y. Liu, B. Schmidt, and D. L. Maskell, "CUDASW++2.0: Enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions," *BMC Res. Notes*, vol. 3, p. 93, 2010.

[20] T. B. Preusser, O. Knodel, and R. G. Spallek, "Short-read mapping by a systolic custom FPGA computation," in *Proc. IEEE 20th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2012, pp. 169–176.

[21] W. Tang, W. Wang, B. Duan, C. Zhang, G. Tan, P. Zhang, and N. Sun, "Accelerating millions of short reads mapping on a heterogeneous architecture with FPGA accelerator," in *Proc. IEEE 20th Annu. Int. Symp. Field-Programmable Custom Comput. Mach.*, 2012, pp. 184–187.

[22] A. Mahram and M. C. Herbordt, "FMSA: FPGA-Accelerated clustaIW-based multiple sequence alignment through pipelined prefiltering," in *Proc. IEEE 20th Annu. Int. Symp. Field-Programmable Custom Comput., Mach.*, 2012, pp. 177–183.

[23] S. Lloyd and Q. O Snell, "Accelerated large-scale multiple sequence alignment," *BMC Bioinformat.*, vol. 12, p. 466, 2011.

[24] Y. Chen, B. Schmidt, and D. L. Maskell, "Accelerating short read mapping on an FPGA (abstract only)," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, Monterey, California CA, USA, 2012, p. 265.

[25] P. Zhang, G. Tan, and G. R. Gao, "Implementation of the Smith-Waterman algorithm on a reconfigurable supercomputing platform," in *Proc. 1st Int. Workshop High-Performance Reconfigurable Comput. Technol. Appl.*, Reno, NV, USA, , 2007, pp. 39–48.

[26] Q. Wang and V. K. Prasanna, "Multi-core architecture on FPGA for large dictionary string matching," in *Proc. 17th IEEE Symp. Field Programmable Custom Comput. Mach.*, 2009, pp. 96–103.

[27] I. Sourdis and D. Pnevmatikatos, "Fast, large-scale string match for a 10 Gbps FPGA-based network intrusion detection system,'' in *Proc. 13th Int. Conf. Field Programmable Logic Appl.*, 2003, pp. 880–889.

[28] T. V. Court and M. C. Herbordt, "Families of FPGA-based algorithms for approximate string matching," in *Proc. 15th IEEE Int. Conf. Appl.-Specific Syst., Archit. Process.*, 2004, pp. 354–364.

[29] M. Aldwairi, T. Conte, and P. Franzon, "Configurable string matching hardware for speeding up intrusion detection," *ACM SIGARCH Comput. Archit. News*, vol. 33, no. 1, pp. 99–107, 2005.

[30] D. Pao, W. Lin, and B. Liu, "Pipelined architecture for multi-string matching," *IEEE Comput. Archit. Lett.*, vol. 7, no. 2, pp. 33–36, Jul.–Dec. 2008.

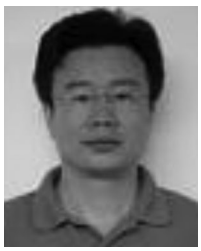
[31] Z. K. Baker and V. K. Prasanna, "A computationally efficient engine for flexible intrusion detection," *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 13, no. 10, pp. 1179–1189, Oct. 2005.
[32] Z. K. Baker and V. K. Prasanna, "Automatic synthesis of efficient intrusion detection systems on FPGAs," *IEEE Trans. Dependable Secure Comput.*, vol. 3, no. 4, pp. 289–300, Oct.–Nov. 2006.
[33] V. Dimopoulos, I. Papaefstathiou, and D. Pnevmatikatos, "A memory-efficient reconfigurable aho-corasick FSM implementation for intrusion detection systems," in *Proc. Int. Conf. Embedded Comput. Syst.: Archit., Model. Simul.*, 2007, pp. 186–193.
[34] A. V. Aho and M. J. Corasick, "Efficient string matching: An aid to bibliographic search," *Commun. ACM*, vol. 18, no. 6, pp. 333–340, 1975.
[35] D. Pao, W. Lin, and B. Liu, "A memory-efficient pipelined implementation of the aho-corasick string-matching algorithm," *ACM Trans. Archit. Code Optim.*, vol. 7, no. 2, pp. 1–27, 2010.
[36] C. Wang, X. Li, X. Zhou, J. Martin, and R. C. C. Cheung, "Genome sequencing using mapreduce on FPGA with multiple hardware accelerators," in *Proc. ACM/SIGDA Int. Symp. Field Programmable Gate Arrays*, 2013, p. 266.
[37] C. Wang, X. Li, X. Zhou, Y. Chen, and R. C. C. Cheung, "Big data genome sequencing on Zynq based clusters in 2014," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, 2014, p. 247.
[38] P. Chen, C. Wang, X. Li, and X. Zhou, "Accelerating the next generation long read mapping with the FPGA-based system," *IEEE/ACM Trans. Comput. Biol. Bioinformat.*, vol. PP, no. 99, p. 1, 2014.
[39] P. Chen, C. Wang, X. Li, and X. Zhou, "Hardware acceleration for the banded Smith-Waterman algorithm with the cycled systolic array, in *Proc. Int. Conf. Field-Programmable Technol.*, 2013, pp. 480–481.
[40] D. Dai, X. Li, C. Wang, and X. Zhou, "Cloud based short read mapping service," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2012, pp. 601–604.
[41] Apache. HBase - Apache HBase Home. (2014). [Online]. Available: http://hbase.apache.org/
[42] Apache. Hadoop. (2014). [Online]. Available: http://hadoop.apache.org/
[43] HDFS Architecture Guide. (2014). [Online]. Available: http://hadoop.apache.org/docs/r1.2.1/hdfs_design.html
[44] H. Li and R. Durbin, "Fast and accurate long-read alignment with Burrows-Wheeler transform," *Bioinformatics*, vol. 26, no. 5, pp. 589–595, 2010.
[45] H. Li, J. Ruan, and R. Durbin, "Mapping short DNA sequencing reads and calling variants using mapping quality scores," *Genome Res.*, vol. 18, no. 11, pp. 1851–1858, 2008.
[46] B. Harris, A. C. Jacob, J. M. Lancaster, J. Buhler, and R. D. Chamberlain, "A banded smith-waterman FPGA accelerator for mercury BLASTP," in *Proc. Int. Conf. Field Programmable Logic Appl.*, 2007, pp. 765–769.
[47] X. Guo, H. Wang, and V. Devabhaktuni, "A systolic array-based FPGA parallel architecture for the BLAST algorithm," *ISRN Bioinformat.*, vol. 2012, p. 11, Article ID 195658, 2012.

Chao Wang (M'11) received the BS and PhD degrees from the University of Science and Technology of China, in 2006 and 2011, respectively, both in computer science. He is an associate professor with Embedded System Lab in the Suzhou Institute of University of Science and Technology of China, Suzhou, China. His research interests include big data and reconfigurable computing. He is now an editor board member of IET CDT MICPRO, IJHPSA. He also serves as the publicity cochair of the ISPA 2014 and HiPEAC 2015. He is a member of the IEEE, ACM, and CCF.

Xi Li is a professor and the vice dean in the School of Software Engineering, University of Science and Technology of China, where he directs the research programs in Embedded System Lab, examining various aspects of embedded system with the focus on performance, availability, flexibility, and energy efficiency. He has lead several national key projects of China, several national 863 projects and NSFC projects. He is a member of the IEEE, ACM, and also a senior member of the CCF.

Peng Chen received the BS degree in computer science from the University of Science and Technology of China in 2010. He is currently working toward the PhD degree in the Embedded System Lab in Suzhou Institute of University of Science and Technology of China, Suzhou, China. His research interests include multiprocessor system on chip, reconfigurable computing, and big data oriented heterogeneous platforms. He is a student member of the IEEE and CCF.

Aili Wang is a lecture in the School of Software Engineering, University of Science and Technology of China. She serves as the guest editor of *Applied Soft Computing*, and *International Journal of Parallel Programming*. She is also a reviewer for the *International Journal of Electronics*. She has published more than 10 International journal and conference articles in the areas of software engineering, operating systems, and distributed computing systems.



Xuehai Zhou is the executive dean in School of Software Engineering, University of Science and Technology of China, and a professor in the School of Computer Science. He serves as a general secretary of steering committee of Computer College Fundamental Lessons, and technical committee of Open Systems, China Computer Federation. He has lead many national 863 projects and NSFC projects. He has published more 100 International journal and conference articles in the areas of software engineering, operating systems, and distributed computing systems. He is a member of the IEEE and ACM, and also a senior member of the CCF.

Hong Yu received the BS degree from Tsinghua University in 2006 and the PhD degree in bioinformatics from the Institute of Genetics and Developmental Biology, Chinese Academy of Sciences, in 2011. He is an assistant professor with the National Center for Plant Gene Research, Institute of Genetics and Developmental Biology, Chinese Academy of Sciences. He has published more than 10 International journal and conference articles in the research areas of bioinformatics on scientific journals, including nature, genome research, bioinformatics, and BMC systems biology.