FPGA Acceleration of Short Read Mapping based on Sort and Parallel Comparison

Youkou Sogabe
Systems and Information Engineering
University of Tsukuba
1-1-1 Ten-ou-dai Tsukuba Ibaraki 305-8573 JAPAN
email: sogabe@darwin.esys.tsukuba.ac.jp

Tsutomu Maruyama
Systems and Information Engineering
University of Tsukuba
1-1-1 Ten-ou-dai Tsukuba Ibaraki 305-8573 JAPAN
email: maruyama@darwin.esys.tsukuba.ac.jp

Abstract—Short read mapping is a process to align the short reads, which are fixed-length fragments of the target genome, to a given reference genome to identify the mutations in the target genome. Because of the rapid development of Next Generation Sequencing (NGS) technologies, faster short read mapping is required. In this paper, we propose an FPGA system for the short read mapping based on sort and parallel comparison of seeds. Seeds are fixed-length sub-strings in the short reads used to map the short read to the reference genome efficiently. In our system, (1) seeds are sorted using bucket sort, and the seeds in a bucket are compared in parallel with the candidate locations on the reference genome, (2) in this comparison, one nucleotide substitution, insertion, and deletion are allowed to achieve higher mapping rate, and (3) two stage search by reconfiguration is used to achieve higher performance. The mapping rate and search time by this approach outperforms existing systems.

I. INTRODUCTION

DNA sequencing is a process to determine the precise sequence of nucleotides of the target genome. To determine the sequence, first, the target genome is randomly fragmented into the fixed-length sub-sequences, called short reads, and then, the short reads are mapped onto a given reference genome, which is a complete genetic sequence that was constructed in advance. In this mapping, genetic variations are allowed, and the sequence of the target genome can be obtained by reconnecting the short reads using their mapping results. By comparing the obtained sequence with the reference genome, the mutations in the target genome can be found. Owing to the rapid growth of the Next Generation Sequencing (NGS) technologies, faster and faster short read mapping is required. Several software tools for the mapping such as BOWTIE[1], BWA [2], and BFAST[3] have been developed.

To accelerate the performance of the short read mapping, several hardware systems have been proposed. In [4], an FPGA aligner based on the BFAST was implemented. The final system with 8 Virtex-6 FPGAs achieved two and one orders of magnitude speedup against BFAST and BOWTIE respectively. In [5], a hybrid approach was proposed, and the alignment and the seed generations were implemented on an FPGA. Its speedup over BWA is about 5, and 70% of the computation time is consumed by the host computer. In [6], an FPGA system for a variant of the FM-index algorithm was implemented on Virtex-6 FPGA, and it showed similar performance with [4].

In [7], we proposed a method to accelerate a hash-index method. In this system, one nucleotide substitution was allowed in the comparison, and its matching rate was considerably higher than previous works, though its processing speed was almost half of [4]. In this paper, we extend this approach to achieve higher processing speed while maintaining the high matching ratio: (1) one nucleotide insertion and deletion are introduced in the comparison as well as the substitution, and (2) the number of comparisons is reduced without degrading the matching ratio, and (3) two stage search by reconfiguration is used to achieve high processing speed and high matching rate. The search in each stage consists of two parts: finding the candidate locations and calculating their scores using Smith-Waterman algorithm[8]. In this paper, we focus on how to find the candidate locations efficiently.

II. SHORT READ MAPPING BY HASH INDEX METHOD

We aim to accelerate the hash-index method used in BFAST[3]. In this method, shorter fixed-length sub-sequences of nucleotides, called seeds, are used as the indexes to identify the locations of the short reads on the reference genome. The seed is further divided into two parts: “index” and “key”, and the locations are searched using two tables: index table, and CAL (candidate alignment location) table. In the following, we describe the details of search using an example shown in Fig.1.

In the following discussion, let the length of short read, seed, index, and key be $l_r$, $l_s$, $l_i$, and $l_k$ respectively ($l_i=l_i+k_k$). In Fig.1, the reference genome is “ACGTAACGTAGC”, and its length is 12. Let $l_i$ be 4. Then, there exist 9 seeds ($l_r=l_s+1$) in the reference genome. Each seed is divided into index ($l_i=2$) and key($l_k=2$), and the seeds with the same index are grouped. Then, their keys and locations on the reference genome are stored in the CAL table. The position of each group on the CAL table is registered in the index table with the number of keys in the group. Suppose that a short read “CGTAATG” is given. Then, there exist $l_r=l_s+1$ seeds in it. For each seed, its location on the reference genome is looked up using the tables. Then, the candidate locations 1 and 6 are obtained, and the short read is compared with the reference genome starting from the locations, and 1 is chosen as the final location. According to [3], when $l_r=22$, more than 80% seeds has only one candidate location, while the number of the seeds that have 10 to more than 100000 candidate locations is more than 15%. Thus, the seeds with more than 8 candidate locations are not registered in the CAL table to reduce the number of candidate locations.
Fig. 1. The index and CAL tables

III. OUR APPROACH

In the original hash-index algorithm, the seeds are processed one by one. In our approach, the seeds are sorted by their indexes, and $P$ seeds of the same index are compared at the same time. $P$ is a parameter decided by the hardware size. Our approach consists of the following steps:

1) Sort: All seeds in the given short reads are extracted, and they are sorted by indexes using bucket sort.

2) Parallel comparison: When $P$ seeds of the same index are obtained, their candidate locations are read from the CAL table in DRAM bank, and compared with the $P$ seeds in parallel.

3) Score calculation: The final matching scores between the short reads and the reference genome are calculated by Smith-Waterman algorithm.

This parallel comparison aims to reduce the computation time of key comparison, and the idle time caused by DRAM access delay by reducing the number of DRAM accesses to $1/P$.

Other features of our approach are:

1) in the key comparison, one nucleotide substitution, insertion, and deletion are allowed,

2) the number of the candidate locations in the CAL table are reduced while maintaining high matching ratio, and

3) two stage search by reconfiguration is used to achieve higher performance.

In the following discussion, we call the matching that allows one nucleotide mutation described above ‘flexible matching’, and the matching without mutation ‘exact matching’.

Fig. 2 shows a block diagram of our architecture. Short reads and their IDs are given to the sort module from the host computer, and all seeds in the short reads are sorted using bucket sort. These short reads and their IDs are also given to the Smith-Waterman module, and are buffered then stored in DRAM memory banks. In the key matching module, the candidate locations for each seed are looked up, and they are given to the Smith-Waterman module through a buffer to calculate their matching scores using Smith-Waterman aligners. The best matching score and its location so far are registered in the DRAM memory banks, and after all matching scores are calculated, they are output to the host computer. In this architecture, the sorting and key matching modules run sequentially, and the Smith-Waterman module runs in parallel with these two modules. In this paper, we focus on the sort and matching modules, because many sophisticated Smith-Waterman aligners have been proposed in many papers, and their processing speed is fast enough.

A. Sorting

Fig. 3 shows the bucket sort unit. First, a short read is set to a shift register. The short read is shifted to left on this shift register controlled by the counter, and all seeds are extracted. The keys of the seeds are once buffered in the internal (small) buckets on FPGA with their position in the short read (counter’s value), and its short read ID. Then, they are moved to the external (large) buckets in DRAM banks by burst-write when the internal buckets become full.

The number of buckets is decided by $l_i$. Here, we introduce a new parameter $L_i$, which is the bit length of the index. Because each nucleotide are encoded to $2b$, $L_i=2^k \times 2$. The total number of internal and external buckets is given by $2L_i$ respectively. When the size of the internal buckets is the multiples of eight, the data in the internal buckets can be moved to their external buckets most efficiently by burst-write. However, when the amount of hardware resources is limited, which is the practical case, the internal bucket size cannot be eight. In the following discussion, we use a parameter $k$ ($k=1, 2, 4, 8$), and the internal bucket size is given by $8/k$.

With this approach, the execution time for sorting all seeds in $N$ short reads are given by

$$T_{sort} = k' \times N_s \times t$$

where $k'=k$ when $k=1$, otherwise $k'=2k$, $N_s$ is the number of the seeds in $N$ short reads ($N_s = N \times (l_i - l_s + 1)$), and $t$ is the clock cycle time.
from the CAL table. Fig. 4 shows an array to compare their keys (called CALs in the following discussion) are read back. At the same time, the candidate locations and their positions in their short reads, and their short read ID) in matched pairs are actually obtained in the second phase. The number of matched keys is counted for each CAL, and the comparison consists of two phases. In the first phase, only the required to evacuate all data in the array.

When an external bucket becomes full, the P seeds (keys, their positions in their short reads, and their short read ID) in it are read back. At the same time, the candidate locations and their keys (called CALs in the following discussion) are read from the CAL table. Fig. 4 shows an array to compare them in parallel. This configuration is chosen to maximize P. The comparison consists of two phases. In the first phase, only the number of matched keys is counted for each CAL, and the matched pairs are actually obtained in the second phase. The execution time of ith first phase is given by

$$T_{1,i} = T_d \times 2 + (c_i + P/4) \times t$$

where $T_d$ is the DRAM access delay, and $c_i$ is the number of CALs used in the ith comparison step. Both index table and CAL table are placed in the DRAM banks, and $T_d \times 2$ is required to obtain the first CAL. $P/4$ is the clock cycles required to evacuate all data in the array.

The execution time of ith 2nd phase is given by

$$T_{2,i} = (m_i + P/4) \times t$$

where $m_i$ is the number of the successful key matching in ith comparison phase.

C. Execution time

In our approach, the Smith-Waterman algorithm is applied to the matched CALs in parallel with the sort and comparison steps. The execution time is decided by the sort and comparison steps, because the number of the matched CALs is not so many, and they can be processed in parallel using several Smith-Waterman aligners. The total execution time is given by

$$T_{total} = T_{sort} + T_{cmp}$$

$$= T_{sort} + N_s/P \times \{T_{1,1} + T_{2,1}\}$$

$$= k \times N_s \times t + N_s/P \times \{T_d \times 2 + ave(c_i) \times t\} + N_s/2 \times t + m_{total} \times t$$

where $ave()$ is the average, and $m_{total}$ is the total number of the key matching. At the end of the matching, the extra time to process the seeds that are left in the buckets is further required.

Its worst-case execution time is almost given by

$$T_{\alpha} = 2^{L_i} \times \{T_d \times 2 + ave(c_{index}) \times t + P/4 \times 2 \times t\}$$

where $ave(c_{index})$ is the average per index.

IV. PERFORMANCE ANALYSIS AND IMPROVEMENT

Here, we consider to tune up the parameters, $k$, $P$, and $ave(c_i)$. For reducing $T_{total}$, smaller $k$, larger $P$, and smaller $ave(c_i)$ are desirable. However, in this decision, we also need to consider the matching accuracy.

1) Two stage matching: $P$ is mainly decided by hardware size. By implementing only the exact matching, larger $P$ can be realized, but its matching accuracy is limited. By implementing the flexible matching, the accuracy can be considerably improved, but this matching requires more hardware resources, and $P$ becomes smaller. To solve this trade-off, two stage search based on reconfiguration is used. In the first stage, only the exact matching is supported to realize largest $P$. In the second stage, the flexible matching is supported, and only the short reads for which their locations could not be found in the first stage are processed.

2) Reducing the number of CALs: In our approach, only the seeds on $s \times i$th position ($i$ is an arbitrary integer) on the reference genome are registered. When the seed has more than eight CALs, the nearest one that has less than or equal to eight CALs is registered instead of it. With this method, the number of CALs can be reduced to almost $1/s$. Fig. 5 shows the matching rate when $s=15$ and $L_i=16$. In this graph, three sets of three lines are shown. The set tagged “0-10 substitutions” shows the matching rate when only 0 to 10 substitutions (the horizontal axis) are added to each short read. In the same way, only insertions or deletions are added in the other two sets. The three lines in each set denote BFAST, our exact matching, and flexible matching. As shown in this graph, when the short reads include mutations, the matching rate with only exact matching becomes considerably worse than BFAST. This accuracy, however, can be improved by using the flexible matching, and it shows better accuracy than BFAST.

3) Performance simulation for deciding $L_i$: To fix the value of $L_i$, we evaluated the performance when $L_i$ is changed using a hardware simulator. Fig. 6 shows the execution time to map 50M short reads in SRR3857731 onto reference genome NCBI v37, when 960 block RAMs are used for the internal buckets, $s=15$, and the flexible matching is supported. The operational frequency is 200MHz. The execution time does
not include \( T_{ex} \), which becomes 2.3 sec in the worst case under this configuration. The best \( L_i \) considerably varies depending on \( P \). In our experiments, we fixed \( L_i \) to 16.

V. EXPERIMENTAL RESULTS

We designed two types of circuits using Verilog HDL and targeted it to Virtex-7 XC7VX690T and Virtex-6 XC6VLX240T. The first type of circuit is for the exact matching, and the second type is for the flexible matching. Vivado ver.2013.4 and ISE 13.4 were used to compile the circuits for XC7VX690T and XC6VLX240T respectively. Table I shows the summary of the resource usage of the four circuits. In Table I, only half of LUTs are used in all circuits. This is to leave LUTs for the Smith-Waterman aligners.

Table II compares the execution time \( t \), matching rate \( \text{rate} \), and speedup over BOWTIE of several software programs, an FPGA system[4], which is one of the fastest FPGA systems, and our circuits when 50M 100bp short reads are mapped (the same set as above is used). The software programs were executed on Intel Core i7 860 (2.8 GHz) with 8GB main memory by 8 threads. In our system, 200Hz is used as the operational frequency to synchronize the DRAM interface. As shown in Table II, the execution time of our ‘exact matching’ is faster than other systems, but its matching rate is a bit worse than BFAST and Olson’s. Our matching rate can be improved by using the flexible matching, but it requires more computation time. The execution time of flexible matching is faster than all other systems when XC7VX690T is used, but slower than Olson’s when XC6VLX240T is used.

By using two stage matching (the exact and flexible matching) by reconfiguration, it becomes possible to achieve high processing speed. The processing speed depends on the mapping rate of exact matching, namely the mutation rates in the short reads. Table III shows the execution time (‘\( t \)’) and matching rate (‘rate’) by this two stage search when the error rates of each nucleotide in short reads is 2% and 4%. By using two stage approach (‘reconfigured’), the processing speed can be improved by 31 to 48% while attaining the high matching rate of the flexible search, when the error rate is 2%.

VI. CONCLUSION

In this paper, we showed that the short read mapping can be accelerated by sorting the seeds in the short reads using bucket sort, and comparing their keys in parallel with the keys of their candidate locations. For this acceleration, the number of candidate locations is reduced without degrading the matching accuracy by introducing one nucleotide substitution, insertion, and deletion in the key comparison. The mapping speed can be further accelerated by introducing the two stage search, which is realized by reconfiguration. In our current implementation, the Smith-Waterman aligners are not attached yet, and their performances are evaluated by a hardware simulator. The processing speed can be further improved by changing the index length adaptively. These are our future work.

REFERENCES