FPGA-Accelerator for DNA Sequence Alignment Based on an Efficient Data-Dependent Memory Access Scheme

Hasitha Muthumala Waidyasooriya, Masanori Hariyama and Michitaka Kameyama
Graduate School of Information Sciences, Tohoku University
Aoba 6-6-05, Aramaki, Aoba, Sendai, Miyagi, 980-8579, Japan
Email: {hasitha, hariyama, kameyama}@ecei.tohoku.ac.jp

ABSTRACT
The mapping of millions of short DNA fragments to a large genome is a very important aspect of the modern computational biology. However, software-based DNA sequence mapping takes many days to complete. This paper proposes an FPGA-based hardware accelerator to increase the mapping speed. We apply a data encoding scheme that reduces the genome size by 90%, and propose a hardware decoder to decode the data in single clock cycle. We also design customized data paths to increase the speed for random data access. According to the experimental results, the speed-up is 15 times compared to its equivalent software application.

Keywords
Short-read mapping, genome sequence alignment, Burrows-Wheeler alignment, FPGA.

1. INTRODUCTION
DNA sequence mapping is an extremely important aspect of modern computational biology. Figure 1 shows the mapping process. It uses a reference genome and short DNA fragments called short-reads. Short-reads are aligned along the reference genome considering the fact that the genomes differ only slightly. Although the mapping process looks simple, it is a very difficult task due to the large size of the genome. Usually, a human genome is large as 3 billion symbols and require billions of short-reads to map it. Therefore, the existing software applications such as Maq [1], BPGA [4], Bowtie [2] and BWA [3] require days or weeks to map a whole genome.

One major problem in software applications is the slow memory access. Especially, when the access pattern is random, the memory access speed drops considerably. Unfortunately, in CPUs, the data paths are fixed and the cache memory does not help much when the amount of data is huge and the access patterns are data dependent. This problem can be solved effectively by designing custom data-paths. FPGAs contain millions of programmable logic gates and are connected to large memories such as 4GB DDR3. In this paper, we propose an FPGA-based architecture to accelerate the short-read mapping using “Burrows-Wheeler alignment (BWA)” [3]. We choose BWA since it is one of the fastest mapping tool among software methods and it provides a massive parallelism.

This paper is an extension of the work done in [6]. In [6], the basic architecture for short-read mapping is given and estimated speed-up is discussed. In this paper, we explain the FPGA-based accelerator architecture in detail. We implemented it on an FPGA and performed the evaluation using real human genome data. We mapped over 200,000 short reads of 90 characters long and discuss the measured results. According to the experiments, 15 times speed-up is observed compared to software-based approach.

2. DNA SEQUENCE ALIGNMENT
In this section, we briefly describe the BWA method shown in Algorithm 1. To explain the algorithm, let us consider the example shown in Fig.2. We have a reference DNA sequence X and a short-read W as shown in Fig.2(a). The inputs are C() shown in Fig.2(a), the occurrence array O(...) shown in Fig.2(b) and short-reads. The number of symbols that are lexicographically smaller than a is given by C(a) where a ∈ {A, C, G, T}. The occurrence array is constructed by applying BW transform to the reference genome X. We recommend to refer [3] and [6] for detailed description of the BWA algorithm. BWA algorithm uses the “exact matching” method explained in [7]. According to [7], if a string W is a substring of the string X and k(aW) ≤ l(aW), string aW is also a substring of X where aW equals the string \{a, W\}. The terms k and l, given by Eqs.(1) and (2) respectively, are the lower and upper bounds of the suffix array (SA) interval of X.

\[ k(aW) = C(a) + O(a, k(W)) - 1 \]  \hspace{1cm} (1)
\[ l(aW) = C(a) + O(a, l(W)) \]  \hspace{1cm} (2)

Note that, the suffix array shown in Fig.2(b) shows the corresponding positions of the elements after the BW transformation to the reference genome.

The input data for the short-read mapping are created using the reference genome. In practical problems, the same
Algorithm 1: Short-read mapping algorithm

begin
  if i < 0 then
    return [k, l]
  end
  I = \phi
  I = I \cup \text{InexRecur}(W, i - 1, z - 1, k, l)
  for each \(a \in \{A, C, G, T\}\) do
    \(k_a = C(a) + O(a, k - 1) + 1\)
    \(l_a = C(a) + O(a, l)\)
    if \(k_a \leq l_a\) then
      \(I = I \cup \text{InexRecur}(W, i - 1, z - 1, k_a, l_a)\)
      if \(a = W[i]\) then
        \(I = I \cup \text{InexRecur}(W, i - 1, z, k_a, l_a)\)
      else
        \(I = I \cup \text{InexRecur}(W, i - 1, z - 1, k_a, l_a)\)
    end
  end
  return I
end

Figure 2: Mapping example

(a) Reference sequence \(X\), short-read \(W\) and \(C(a)\) of \(X\)

(b) Occurrence array of \(X\)

3. ACCELERATOR ARCHITECTURE

3.1 Overall architecture

The overall architecture of the accelerator is shown in Fig.3. It consists of two DDR3 SDRAMs, a memory controller, and two groups of PEs belong to channel 1 and channel 2. A channel contains 32 PEs. The parallel data processing is achieved by executing different short-reads in parallel on 64 PEs. The occurrence array and short-read data are transferred to the DDR3 memory.

Structure of a PE is given in Fig.4. It consists of a 32-bit adder, a comparator and pipeline registers to perform the calculations explained in algorithm 1. After finishing one “InxRecur” procedure, a new one is loaded from the register file. In each “InxRecur” procedure, new calls to the same procedure are generated as explained in Algorithm 1. The parameters of such recursive calls are stored in the register file, so that we can keep a track of all the recursive calls.

3.2 Data encoding

One common problem of the BWA algorithm is the enormous amount of data. To explain this, let us refer the occurrence array example in Fig.2(b). Our task is to store the occurrence array data. There are total of seven entries from 0 to 6. Each entry gives the number of “A, C, G, T” symbols. Note that, “$” which represents the end of the reference genome is stored separately. Considering the worst case where all the symbols are the same, we need 3 bits each to represent the number of “A, C, G, T” symbols. Therefore, a total of 12 bits are required to store one entry and 72 bits for all 6 entries. Applying this calculation, to store the occurrence array of a genome as large as human, we need 48 GB of data. We could rarely find an FPGA that can hold such a huge amount of data. To solve this problem, we encode the occurrence array data, and build a hardware decoder to decode any entry in a single clock cycle.

Figure 5 shows the encoded data of the occurrence array shown in Fig.2(b). The first two least significant bits rep-
Figure 5: Encoded occurrence array

<table>
<thead>
<tr>
<th>Bit number</th>
<th>21:10</th>
<th>9:8</th>
<th>7:6</th>
<th>5:4</th>
<th>3:2</th>
<th>1:0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entry</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Number of</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Symbol</td>
<td>T</td>
<td>G</td>
<td>C</td>
<td>A</td>
<td>T</td>
<td>C</td>
</tr>
<tr>
<td>Code</td>
<td>001</td>
<td>010</td>
<td>010</td>
<td>001</td>
<td>010</td>
<td>10</td>
</tr>
</tbody>
</table>

Figure 6: Hardware decoder for occurrence of “T”

Figure 7: DDR3 memory access

3.3 Fast memory access using multiple address streams

In the proposed architecture, multiple PEs access the same DDR3 memory. The memory addresses accessed by PEs depend on the input data. Therefore, we cannot predict the memory access patterns efficiently and it is difficult to optimize the memory access off-line. In such problems, arbiters are used to select a single request in each clock cycle. The designed data path for the memory access is shown in Fig.7. Multiple PEs send address requests in parallel to the arbiter. The arbiter allows one request to proceed in each clock cycle so that FIFO is filled with multiple addresses. Those addresses are sent one-by-one to access the memory.

This method works well if both the accelerator and DDR3 controller clock frequencies are the same. However, the DDR3 controller clock frequency is much larger (200MHz) than that of the accelerator’s (80MHz). Therefore, the data access rate always decided by the slower clock frequency, irrespective of how fast the DDR3 controller clock is. Note that, the DDR3 has a 64-bit data path with a 800MHz clock. In the DDR3 controller, the clock frequency is 200MHz and the data-path width is 512 bits so that it matches the data rate of the DDR3 memory. Since the accelerator clock is slower, valid address requests arrive slower than they are read by the faster DDR3 controller. Therefore, as shown in Fig.8(a), a few valid addresses are present in the FIFO and that reduces the access speed. To solve this problem, we use two data streams in parallel as shown in Fig.8(b). Two streams write two 32-bit addresses to the FIFO simultaneously in each accelerator clock cycle. Note that the FIFO is designed to have 64-bit data-path for write operation while 32-bit data-path for the read operation. Since the DDR3 controller clock is faster than the accelerator clock, more valid address requests are sent to the DDR3 controller. This increases the memory access speed.

4. Evaluation

For the evaluation, we used DE5 board [8] that contains “Altera Stratix V 5SGXEA7N2F45C2 FPGA” and two 4GB DDR3-SDRAMs. The system contains a core i7-3960x CPU and a DE5 board connected through the PCI express port. The operating frequency of the accelerator is 80MHz. Table 1 shows the resource usage. The FPGA accelerator uses 60% of the look-up-tables (LUTs) in the FPGA. The most of the resources are used by the PE array while just 5% of the LUTs are used to design the PCI express and DDR3 controllers. Therefore, we can increase the number of PEs to reduce the processing time further.

Table 2 shows the memory bandwidth comparison of the CPU-based and the FPGA-based systems. Although CPU has a better theoretical bandwidth, it reduces drastically for random data access. Since the memory access in short-read mapping is random, small bandwidth is a big problem in CPU-based accelerations. However, in FPGA, we design custom data paths and address generation units for random access. In random access, the memory access speed of FPGA is 10 times larger compared to that of a CPU.

We mapped 200,000 short reads of a human genome using the FPGA accelerator. Each short-read is 90 symbols...
long. Table 3 shows the measured processing time. For the comparison, we used a software-based system that contains an Intel Xeon E5-2643 3.3GHz processor and CentOS 6.3 operating system. The software is written in C language and compiled using gcc compiler. According to the results, the speed-up is 15 ∼ 30 times. The processing time increases exponentially when the number of misses (SNP and indels) is over three. When the number of misses are small, the software-based processing contains a relatively large portion of hard disk access time. However, when the number of misses is greater than three, the mapping time is so large that the time required to access the hard disk is insignificant. Therefore, the speed-up drops to 15 times. In real world problems of genome mapping, the maximum number of misses allowed is three to four. Therefore, we can say that the proposed accelerator has a 15 times speed-up compared to software. As a result, a whole genome can be mapped within few hours. This measured speed-up of 15 times using real genome data is larger than the estimated speed-up of 10 times in our previous work [6].

5. CONCLUSION

We have proposed a hardware accelerator architecture for DNA sequence mapping. We successfully implemented the proposed architecture on an FPGA and mapped 200,000 short-reads. The measured speed-up is 15 to 30 times compared to the equivalent software application. It is possible to increase the processing power by choosing latest FPGAs such as Altera Stratix V with more LUTs and memory. Moreover, we can use multiple FPGAs connected by fiber optics to increase the processing speed massively. Therefore, the proposed FPGA accelerator has a great potential to dramatically increase the processing speed.

Acknowledgment

This work is supported by MEXT KAKENHI Grant Number 24300013.

6. REFERENCES