Pacific Symposium on Biocomputing 7:271-282 (2002)

High Speed Homology Search with FPGAs*

Yoshiki YAMAGUCHI, Tsutomu MARUYAMA
Institute of Engineering Mechanics and Systems, University of Tsukuba,
1-1-1 Ten-ou-dai Tsukuba Ibaraki, 305-8573, JAPAN

Akihiko KONAGAYA
Japan Advanced Institute of Science and Technology,
1-1 Asahidai Tatsunokuchi Ishikawa, 923-1292, JAPAN
Japan Riken Genomic Sciences Center,
1-7-22 Suehiro Tsurumi Yokohama Kanagawa, 230-0045, JAPAN

We will introduce a way how we can achieve high speed homology search by only adding one off-the-shelf PCI board with one Field Programmable Gate Array (FPGA) to a Pentium based computer system in use. FPGA is a reconfigurable device, and any kind of circuits, such as pattern matching program, can be realized in a moment. The performance is almost proportional to the size of FPGA which is used in the system, and FPGAs are becoming larger and larger following Moore's law. We can easily obtain latest/larger FPGAs in the form off-the-shelf PCI boards with FPGAs, at low costs. The result which we obtained is as follows. The performance is most comparable with small to middle class dedicated hardware systems when we use a board with one of the latest FPGAs and the performance can be furthermore accelerated by using more number of FPGA boards. The time for comparing a query sequence of 2,048 elements with a database sequence of 64 million elements by the Smith-Waterman algorithm is about 34 sec, which is about 330 times faster than a desktop computer with a 1GHz PentiumIII. We can also accelerate the performance of a laptop computer using a PC card with one smaller FPGA. The time for comparing a query sequence (1,024) with the database sequence (64 million) is about 185 sec, which is about 30 times faster than the desktop computer.

1 Introduction

In the past several years, there has been a rapid increase in genetic and genomic database, and the pattern matching problems in bioinformatics require huge time for the computations. Many algorithms[4,5,6] and dedicated hardware systems[11,12,13] have been developed. The result obtained there is a trade-off of quality, time and cost. With desktop computer systems, it is unrealistic to check all pattern matching possibilities within a reasonable time. Therefore, simplified (but still very effective) algorithms have been designed and used on the systems. With dedicated hardware systems, the computation time can be

This work was supported by Grant-in-Aid for Scientific Research on Priority Areas (C) "Genome Information Science" from the Ministry of Education, Culture, Sports, Science and Technology of Japan, and Japan Society for the Promotion of Science (JSPS) Research Fellowships for Young Scientists (#5304).

drastically improved, and all the possibilities can be checked, because most of the pattern matching problems have many parallelism in them. However, the cost of the systems are very expensive.

Field Programmable Gate Array (FPGA) is a reconfigurable device designed for rapid prototyping, and any kinds of circuits can be realized on the FPGA in a moment by downloading configuration data from host computers or dedicated memories. The performance is almost proportional to the size of the FPGA because the parallelism of computation is limited by the size. FPGAs are becoming larger and larger following Moore's low (the number of transistors in a fixed size (namely the size of FPGAs) become twice in every 18 months). We can easily obtain the latest/largest FPGAs in the form off-the-shelf PCI boards with FPGAs, which are now being shipped from many companies [7], and we can obtain many kinds of these boards at low cost. FPGAs begin to be used as accelerators in many application areas [8,9,10] and also used in some dedicated hardware systems [11,13] for bioinformatics.

We will show that we can achieve high performance in homology search by only adding one off-the-shelf FPGA board to a Pentium based computer system in use. The performance can be furthermore accelerated by using more number of FPGA boards. In our approach, the search is divided into two phases because the FPGAs do not have enough hardware resources for the pattern matching problems in bioinformatics. Therefore, different configuration data (namely different circuits) are downloaded from the host computer in each phase in order to make up for the limited hardware resources. The configuration data can be easily modified for new FPGA boards, because they are generated from the programs written in hardware description languages without assuming any special hardware resources on the FPGA boards.

This paper is organized as follows. Section 2 describes the overview of our approach, and the details of the approach is given in section 3. Then, experimental results of the approach are shown in section 4. In section 5, current status and future works are given.

2 Overview of the Approach

Our current target problems are homology search problems, and the Smith-Waterman algorithm[1] is used in all comparison between query sequences and database sequences. In this section, we describe the overview of our approach.

2.1 Hardware and Software for our Approach

We need the followings as components of the hardware platform (Figure 1).

1. one off-the-shelf FPGA board (with PCI bus interface), and
2. one host computer (a Pentium based computer, because driver programs for most FPGA boards run only under WINDOWS or LINUX).

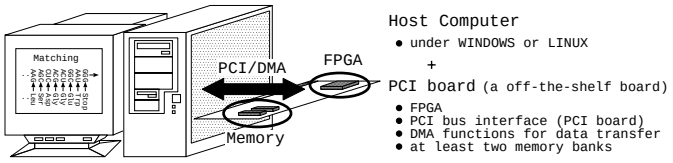

Figure 1: Required Components of Hardware Platform

The softwares which are necessary for our approach are

1. drivers programs to control FPGA boards from the host computer, which are developed by the board maker and attached to the FPGA boards, and
2. CAD tools for the FPGA, only if we need to modify the configuration data for new FPGA boards.

Among the components shown above, what we have developed are

1. the programs for the circuits which are implemented on the FPGA, and
2. interface programs which run on the host computer.

In our programs for the circuits, only the two memory banks to transfer data between the FPGA and the host computer are assumed. Most FPGA boards have at least two memory banks in order to receive data from the host computer while the FPGA is running using another memory bank. Therefore, configuration data for new FPGA boards can be easily generated by only changing some parameters in the programs (FPGA size, memory size in FPGA, I/O pin assignment and so on).

The interface programs need to control FPGA boards using the driver programs. The structure of driver programs depends on the boards, and we need to modify a part of the programs for each board.

2.2 Advantage and Disadvantage of the Approach

Before describing the details of our approach, we would like to summarize the advantage and disadvantage of the approach compared with the dedicated hardware systems.

First, the advantages of our approach are as follows.

1. Many kinds of FPGA boards are shipped from many companies[7], and the costs of the boards are relatively low. We can choose FPGA boards according to our requirements and budgets. For example, the cost of our largest FPGA board is several times the cost of a Pentium based desktop computer system, while the cost of PC card with a smaller FPGA is less than a half of the cost of a laptop computer.
2. It is easy to obtain the boards with latest FPGAs (namely larger FPGAs) as soon as the FPGAs are shipped, which is very important because the performance of the approach is almost proportional to the size of FPGAs.
3. It is possible to replace the FPGA board and the host computer independently.
4. By making the configuration data and the programs for the configuration data open, many users can accelerate their search by only purchasing one off-the-shelf FPGA board.

On the other hand, the disadvantages are as follows.

1. In general, off-the-shelf FPGA boards do not have enough hardware resources for homology search. Especially memory size and memory bandwidth are not sufficient. Furthermore, we assume only two memory banks on the board for data transfer between the host computer and the FPGA, and only the internal memory on the FPGA is used for the homology search in order to maintain portability of our circuits. Because of this limited memory size and memory bandwidth,
   (a) Query sequences can not be compared with long database sequences at once. Therefore, query sequences are always compared with subsequences of the database sequences (automatically divided during the search), and results against only the fragments in the subsequences can be shown (the length of the fragments can be specified by users).
   (b) Some parts of the database sequences are processed twice, and the size of the parts is almost proportional to the length of the query sequences. Therefore, the performance becomes worse as the query sequences becomes longer, though it is negligible if we can use large size FPGAs.
   (c) With smaller FPGAs, the length of query sequences is limited (long query sequences can not be processed). For example, with our PC card with one Virtex XCV300, the maximum length of the query sequence is 1024.

2. Software environment is still poor. Its improvement is one of the our future works.

2.3 Scalability

The performance is almost proportional to the number of FPGA boards when

1. the query sequence is compared with many database sequences which are stored in different hard disks, or
2. each database sequence is divided into subsequences which are stored in different hard disks,

because the database sequences or the subsequences can be compared with the query sequence independently. However, the data transfer rate of the PCI bus is limited, and many FPGA boards can not be attached to one host computer. We have not evaluated the performance when we use more than one FPGA board, but according to our estimation, more than two boards should not be attached to one host computer.

By connecting more hardware platforms by Ethernet, we can easily accelerate the performance furthermore. The performance is almost proportional to the number of hardware platforms. Then, the total performance can be comparable with large size dedicated hardware systems.

3 Details of the Approach

In this section, we describe the details of our approach. The features of our approach are :

1. multi-thread computation in order to achieve high performance, and
2. two phase search in order to make up for limited memory bandwidth.

3.1 Parallel Processing of Dynamic Programming

Before describing the details, we would like to introduce the dynamic programming algorithm. As shown in Figure 2, a query sequence and a database sequence are compared inserting gaps. Scores for each matching of the elements and inserting gaps are given by score matrices[2,3]. The computation order of pattern matching by dynamic programming is $m \times n$ when the length of the query sequence and database sequence are m and n respectively. Therefore, it is unrealistic to use dynamic programming algorithm against long database sequences on desktop computer systems.

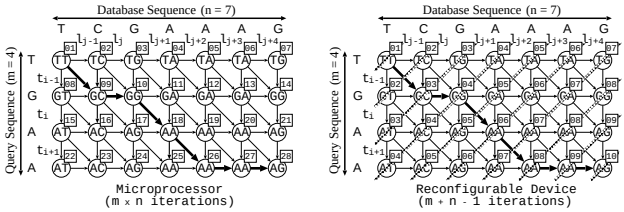

Figure 2: Parallel Processing of Dynamic Programming

With dedicated hardware systems, or reconfigurable devices such as FPGAs, we can process matching of elements in parallel. Figure 2 shows how the matching of the elements is processed in parallel. In the right-hand side part in Figure 2, elements on each diagonal line are processed at once. Therefore, the order of the computation can be reduced to $m+n-1$ from $m \times n$ if m elements can be processed in parallel. If the size of the hardware is not large enough to compare m elements at once, the first p elements (suppose that the hardware process p elements in parallel) of the query sequence are compared with the database sequence at once, and the scores of all p-th elements are stored in temporal memory. Then, the next p elements of the query sequence are compared with the database sequence using the scores stored in the temporal memory.

3.2 Structure of Processing Unit and Multi-thread Computation

Figure 3 shows a structure of our processing unit for dynamic programming. It consists of four stages, and takes four clock cycles to compute scores on each cell on the dynamic programming array (Figure 2). However, by overlapping the computation, we can start to compute the scores of elements on the next diagonal line (Figure 2) in every two clock cycles.

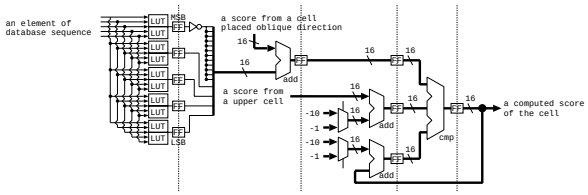

Figure 3: Implementation of a Processing Unit

Figure 4 shows how a database sequence whose length is n, is compared with a query sequence whose length is m. In the figure, each circle represents a processing unit. If the length of the query sequence (m) is not larger than the number of processing elements on the FPGA (p), the query sequence can

be processed at once as shown in Figure 4. In this case, it takes $2 \times (m+n-1)$ cycles to compare the two sequences, because the processing units have to wait for one clock cycle to compare elements on the next diagonal lines as described above, which means that the units are idle for one clock cycle in every two clock cycles.

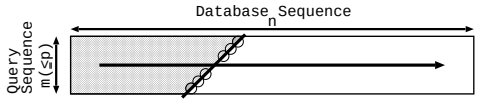

Figure 4: Sequential Execution of Dynamic Programming

Suppose that the length of the query sequence (m) is longer than the number of processing units on the FPGA (p). Then, in the naive approach :

1. first, the first p elements of the query sequence are compared and the intermediate results (all p-th scores on lower edge of upper half in Figure 5(a)) are stored, and
2. then, the next p elements of the query sequence are compared using the intermediate results.

In this case, it takes $2 \times 2 \times (p + n - 1)$ cycles to compare the two sequences, and processing units become idle for one clock cycle in every two clock cycles as described above.

We can reduce the computation time by the multi-thread computation method. In the multi-thread computation :

1. first, p elements on the diagonal line in upper half in Figure 5(b) are processed, and the score of p-th element is stored on temporal registers, and
2. then, the next p elements on the diagonal line in lower half are processed without waiting for one clock cycle using the intermediate result.

By interleaving the processing of elements in upper half and lower half, we can eliminate the idle cycles of the processing elements. The clock cycles become $2 \times (p + n - 1) + 2 \times p$, which is almost equal to $2 \times n$ because n is much longer than p in most cases.

When the length of the query sequence (m) is longer than twice the number of the processing units ($2p$), the multi-thread computation shown in Figure 5(b) is repeated according to the length of the query sequence. In this case, when the first $2p$ elements in the query sequence are processed, scores of all $2p$-th elements are stored in memories (all n scores are stored in total), and used for the computation of the next $2p$ elements.

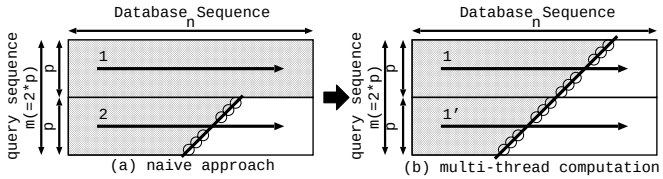

Figure 5: Multi-thread Execution of Dynamic Programming

3.3 Two Phase Search

First Phase

In the first phase, database sequences are divided into sub-sequences, because the size of the intermediate results described above is very large, and can not be stored in the internal memory of the FPGA at once.

Figure 6(a) shows how a long database sequence is compared with the query sequence. The database sequence is divided into sub-sequences of size s (s is decided based on the size of the internal memory of the FPGA). Then, each sub-sequence is compared with the query sequence by the multi-thread method. As shown in Figure 6(a), first, 1 and $1'$ are processed, and then 2 and $2'$ are processed.

In Figure 6(b), in each comparison with the sub-sequences, scores on upper edge (position α_e to α_m) are sampled and compared with scores on lower edge (position β_e to β_m). The score at position α_f is compared with the score at position β_m. The difference of the scores are stored in the two memory banks on the FPGA board, and then sent to the host computer. The host computer sorts the differences, and shows them with the positions on the database sequence. Thus, in our approach, the query sequence is compared with all fragments of the size $k \times l$, and the scores against each fragment by the Smith-Waterman algorithm are shown to the users. The interval of the sampling k and the distance between the two scores $k \times l$ can be specified by users. However, if the k is too small, many data have to be sent to the host computers, and the performance will go down. We assume that the length of the fragment $(k \times l)$ is from twice to four times the query sequence.

In this division to the sub-sequences, some parts whose length is $k \times (l-1)$ are overlapped in order to compare the query sequence with all fragments of length $k \times l$ (Figure 6(a)). These parts are compared twice, and become the major overhead in the first phase. The length of the overlapped area $(k \times (l-1))$ is decided based on the length of the query sequence in general. Therefore, this overhead is almost proportional to the length of the query sequence. This overhead becomes larger as the length of the query sequence becomes longer,

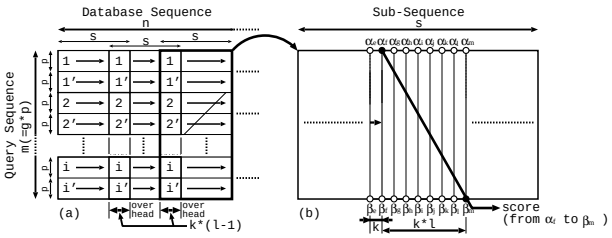

Figure 6: First Phase Execution

and becomes relatively larger as the size of the internal memory of the FPGA becomes smaller (namely the size of the FPGA becomes smaller).

In order to achieve higher performance in the first phase, we need to implement more processing units on the FPGA. The size of the processing unit is proportional to its data width. Therefore, we can implement more units by reducing the data width. However, with narrower data width, the scores may cause overflow or underflow during the comparison with the sub-sequences. In order to avoid the over/underflow, we need to make the size of the sub-sequences smaller too, but this means that more areas have to be overlapped. Therefore, we need to find good balance between the sub-sequence size and the parallelism. In the current FPGAs, the size of the internal memory is not so large, and it gives the best performance when we decide the data width base on only the size of the internal memory size of the FPGA.

Second Phase

In order to display optical alignments, we need to find the path from the upper left position to the lower right position which gives the best score as shown in Figure 2 (in the first phase, only the best score is computed, and all the information about the path is discarded during the computation). We need 2 bits for each cell on the array in Figure 2 to distinguish where the path comes from (from upper, upper left or left). Therefore, the number of elements which can be processed in parallel (namely the performance of the second phase) is decided by the FPGA's data width to the memory banks on the FPGA board, not by the size of the FPGA. If the width is $2 \times p$, p elements can be processed in parallel. In the second phase, only the information about the path is output, because the score is already obtained in the first phase.

However, the number of the fragments that we need to display their alignments, is not so many and the performance of this phase is not so important. If the length of the query sequence is less than a few thousand, we can obtain

the optical alignments against one fragment within 1 sec by a desk computer.

4 Experiments

We have tested the performance of our approach under two environments, a desktop computer with a FPGA board and a laptop computer with a PC card.

4.1 Desktop Environment

One FPGA board (RC1000-PP by Celoxica) [14] is used to evaluate the performance on desktop environment. The board has four memory banks, and two of them are used for data transfer between the FPGA board and the host computer. The FPGA (Xilinx XCV2000E) on the board is one of the largest FPGAs that we can obtain now. We could implement 144 processing elements for the first phase of the homology search, and they run at 40 MHz. The size of the internal memory of the FPGA is 640 Kbits, and the length of the subsequence becomes 32768 elements. Therefore, the overhead caused by the overlapped area becomes about 5 - 10% when the length of the query sequence is 2048 and the size of the fragment is several times of the query sequence.

Figure 7 shows the relation between the time of the first phase and the length of the query sequence, when the length of the database sequence is 64 million. The slope of the search time becomes slightly larger as the length of the query sequence becomes larger, because the percentage of the overhead by the overlapped area will gradually increase. The speedup compared with a PentiumIII 1GHz under LINUX (kernel version 2.2.5 and gcc-2.91.66) is 327 times when the length of the query sequence is 2048.

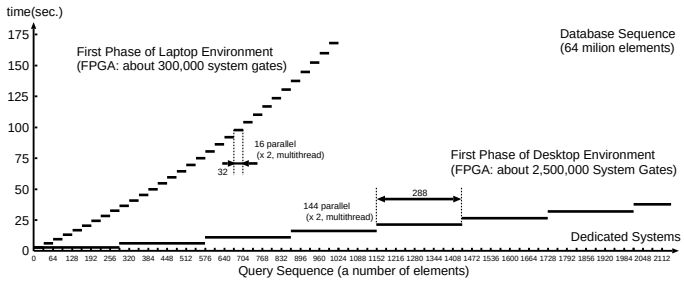

Figure 7: Comparison between Desktop and Laptop Environment

As for the second phase, we can process 32 elements in parallel, because we can write 64 bits to the memory banks on the board at once. The computation time for a query sequence of 2048 elements and a fragment of 8192 elements is about 13 msec. This is about 102 times faster than the PentiumIII 1GHz.

4.2 Laptop Environment

One PC card (Wildcard by Annapolis Micro Systems, Inc.)[15] with one FPGA (XCV300 by XILINX) is used to evaluate the performance on laptop environment. The PC card has two memory banks (32bits width and 256KB per each block), and these banks are used to transfer data between the PC card and the host computer. The size of the FPGA XCV300 is about one seventh of the FPGA XCV2000E.

In this case, we could implement 16 processing units on the FPGA, and they run at 40 MHz. The size of the internal memory of the FPGA is 64 Kbits, and the length of the subsequences becomes 4096 elements. The overhead caused by the overlapped area becomes about 25 to 50% when the length of the query sequence is 1024 and the size of the fragment must be several times of the query sequence. Therefore, we can not compare query sequences longer than 1024.

Figure 7 shows the relation between the time of the first phase and the length of the query sequence, when the length of the database sequence is 64 million. The slope of the search time becomes larger according to the size of the query sequence, because the percentage by the overhead for the overlapped area will increase. The computation time is about 30 times faster than a PentiumIII 1 GHz under LINUX (kernel version 2.2.5 and gcc-2.91.66) when the length of the query sequence is 1024.

As for the second phase, we can process 16 elements in parallel, because we can write 32 bits to the memory bank on the card at once. The computation time for a query sequence of 1024 elements and a fragment of 4096 elements is about 7 msec. This is about 50 times faster than the PentiumIII 1GHz.

5 Current Status and Future Works

We have developed the circuits for homology search, and showed that we can achieve high performance using off-the-shelf FPGA boards. The performance is almost comparable with small to middle class dedicated hardware systems when we use one board with one of the latest FPGAs (Xilinx XCV2000E). The time for comparing a query sequence of 2048 elements with a database sequence of 64 million elements by the Smith-Waterman algorithm is about 34 sec, which is about 330 times faster than a desktop computer with a 1GHz PentiumIII. We can also accelerate the performance of a laptop computer using one PC card with one FPGA (Xilinx XCV300). The time for comparing a query sequence (1024) with database sequence (64 million) is about 185 sec, which is about 30 times faster than the desktop computer.

We are now evaluating the performance for the translated nucleotides. When we need to translate the sequences during the comparison, the size of each unit on the FPGA becomes about 10% larger and the parallelism in the first phase will go down to 120 from 144 (about 20% performance down). We are now improving the circuits of the unit to achieve higher performance.

Some parts of the programs for the homology search are still under development, and we also need to improve other parts. We are also developing softwares for parallel processing of the homology search with more number of pairs of FPGAs and host computers connected by Ethernet.

We are also planning to accelerate other pattern matching problems in bioinformatics with FPGAs.

References

1. Smith T. F. and Waterman M. S.: *Identification of Common Molecular Subsequences*, Journal of Molecular Biology 147, 195-197, (1981).
2. Steven Henikoff and Jorja G. Henikoff: *Amino Acid Substitution Matrices from Protein Blocks*, Proc. Natl. Acad. Sci. 89, 10915-10919, (1992).
3. Jones,D.T., Taylor,W.R. and Thornton,J.M.: *The Rapid Generation of Mutation Data Matrices from Protein Sequences*, CABIOS 8, 275-282, (1992).
4. Altschul,S.F., Gish,W., Miller,W., Myers,E.W. and Lipman,D.J.: *Basic Local Alignment Search Tool*, J. Mol. Biol. 215, 403-410, (1990).
5. Pearson,W.R. and Lipman,D.J.: *FASTA: Improved tools for biological sequence comparison*, Proc. Natl. Acad. Sci. USA 85, 2444-2448, (1988).
6. Altschul,S.F., Madden,T.L., Schaffer,A.A., Zhang,J., Zhang,Z., Miller,W., and Lipman,D.J.: *Gapped BLAST and PSI-BLAST: a new generation of protein database search programs*, Nucl. Aci. Res. 25(17), 3389-3402, (1997).
7. http://www.optimagic.com/boards.html
8. http://www.fccm.org
   (IEEE Symposium on Field-Programmable Custom Computing Machines)
9. http://www.ecs.umass.edu/ece/fpga2002/index.html
   (ACM International Symposium on Field-Programmable Gate Arrays)
10. http://xputers.informatik.uni-kl.de/fpl/index_fpl.html
    (the International Conference on Field-Programmable Logic and Applications)
11. http://www.compugen.com
12. http://www.paracel.com/index.html
13. http://www.timelogic.com
14. http://www.celoxica.com
15. http://www.annapmicro.com
16. http://www.xilinx.com