Multidimensional Dynamic Programming for Homology Search on Distributed Systems

Shingo Masuno[1], Tsutomu Maruyama[1] and Yoshiki Yamaguchi[1]
Akihiko Konagaya[2]

[1] Systems and Information Engineering, University of Tsukuba,
1-1-1 Ten-ou-dai Tsukuba Ibaraki, 305-8573, JAPAN
masuno@darwin.esys.tsukuba.ac.jp
[2] RIKEN Genomic Sciences Center, 1-7-22 Suehiro-cho Tsurumi-ku
Yokohama Kanagawa, 230-0045, Japan

Abstract. Alignment problems in computational biology have been focused recently because of the rapid growth of sequence databases. By computing alignment, we can understand similarity among the sequences. Dynamic programming is a technique to find optimal alignment, but it requires very long computation time. We have shown that dynamic programming for more than two sequences can be efficiently processed on a compact system which consists of an off-the-shelf FPGA board and its host computer (*node*). The performance is, however, not enough for comparing long sequences. In this paper, we describe a computation method for the multidimensional dynamic programming on distributed systems. The method is now being tested using two nodes connected by Ethernet. According to our experiments, it is possible to achieve 5.1 times speedup with 16 nodes, and more speedup can be expected for comparing longer sequences using more number of nodes. The performance is affected only a little by the data transfer delay when comparing long sequences. Therefore, our method can be mapped on any kinds of networks with large delays.

1 Introduction

Alignment problems in computational biology, namely homology search, have been focused recently because of the rapid growth of sequence databases[1–3]. By computing alignment, we can investigate similarity among the sequences. Dynamic programming is a technique to find optimal alignment among sequences. In dynamic programming, all causal connections to the final result are stored, and back-traced in order to obtain the optimal alignment. Its computational complexity, however, is very large (order L^N to compare N sequences of length L), and it is not realistic to use algorithms based on dynamic programming even for alignment between two sequences on desk-top computers. In order to reduce the computation time, many heuristic algorithms[6–8] or hardware systems [9–15] have been proposed. Most of them, however, are designed for two-dimensional alignment (alignment between two sequences) because of the complexity to calculate alignment among more than two sequences under limited hardware resources. We have already proposed computational methods for more than two

sequences [16, 17], and shown that high performance can be achieved on a compact system which consists of an off-the-shelf FPGA board and its host computer (*node*). The performance is, however, not enough for comparing long sequences.

In this paper, we describe a computation method for the multidimensional dynamic programming on distributed systems, which consist of the nodes connected as a ring. The communication pattern between the nodes in our approach is very simple and regular. Each node receives data from its predecessor, and sends its results to its successor. This data transfer can be overlapped with the computation of the dynamic programming. The method is now being tested using two nodes connected by Ethernet.

This paper is organized as follows. Section 2 introduces the outline of dynamic programming for homology search, and our computation method for more than two sequences are described in Section 3. The parallel computation method on distributed systems are given in Section 4, and the estimated performance based on the experimental results is given in Section 5. The current status and future works are given in Section 6.

2 Dynamic Programming for Homology Search

In the dynamic programming for homology search, sequences are compared inserting gaps with extra costs. Figure 1 shows an example of alignment of two sequences by dynamic programming (two-dimensional). In Figure 1(A), scores on each node on the search space ($M \times N$) are calculated using the equation in Figure 2. Scores for each matching between two elements ($\mathrm{M_S}[a[x], b[y]]$) and inserting gaps (GC()) are given by *score matrices* [4, 5]. In each node, there are three candidates of its score (from the left-upper node, upper node and left node) in two-dimensional search, and the maximum of them is chosen. The paths which give the maximum values are stored, and after calculating scores of all nodes, the paths are back-traced from the last node to the start node to obtain the alignment of the two sequences (Figure 1(B)).

To obtain an alignment of more than two sequences, the same procedure is applied to the sequences. The search space of N-dimensional dynamic programming becomes L^N (when N sequences have length L). As indicated by the equations in Figure 2,

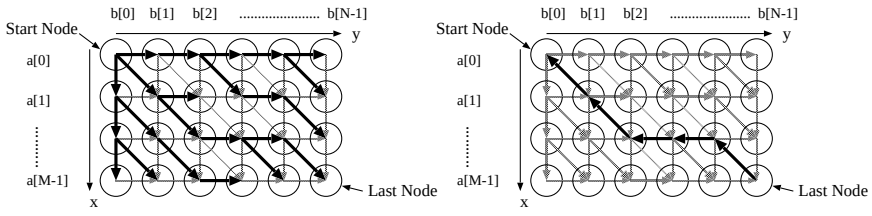


Fig. 1. Two Dimensional Dynamic Programming

Two-Dimensional Search:

$$\text{score}(x, y) = \max \left\{ \begin{array}{l} \text{score}(x\text{-}1, y\text{-}1) + \text{M}_\text{S}[a[x], b[y]] \\ \text{score}(x, y\text{-}1) + \text{GC}(x, \text{-}) \\ \text{score}(x\text{-}1, y) + \text{GC}(\text{-}, y) \end{array} \right\}$$

Three-Dimensional Search:

$$\text{score}(x, y, z) = \max \left\{ \begin{array}{l} \text{score}(x\text{-}1, y\text{-}1, z\text{-}1) + \text{M}_\text{S}[a[x], b[y], c[z]] \\ \text{score}(x, y\text{-}1, z\text{-}1) + \text{M}_\text{S}[\text{-}, b[y], c[z]] + GC(x, \text{-}, \text{-}) \\ \text{score}(x\text{-}1, y, z\text{-}1) + \text{M}_\text{S}[a[x], \text{-}, c[z]] + GC(\text{-}, y, \text{-}) \\ \text{score}(x\text{-}1, y\text{-}1, z) + \text{M}_\text{S}[a[x], b[y], \text{-}] + GC(\text{-}, \text{-}, z) \\ \text{score}(x\text{-}1, y, z) + \text{GC}(\text{-}, y, z) \\ \text{score}(x, y\text{-}1, z) + \text{GC}(x, \text{-}, z) \\ \text{score}(x, y, z\text{-}1) + \text{GC}(x, y, \text{-}) \end{array} \right\}$$

Fig. 2. Equations to calculate Scores

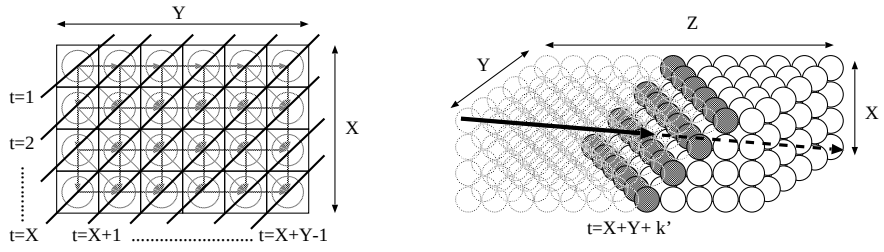
1. the number of candidates of the score for each node is $2^N - 1$ in N-dimensional dynamic programming, and
2. the size of score matrices is k^N (k is the number of type of elements in the sequences), which becomes very large for larger N.

Figure 3 shows the maximum parallelism in dynamic programming. As shown in Figure 3, nodes on a diagonal line (plane) can be processed in parallel. The maximum parallelism in N-dimensional search is the product of the size of N-1 sequences (in the maximum case). When N=2, the maximum parallelism is Y, and it takes $X \times Y$ - 1 steps to calculate the alignment.

3 Multidimensional Dynamic Programming on an FPGA

In the dynamic programming, we need to store paths to each node to backtrace. The total size of the paths becomes L^N(the number of the nodes in the search space) $\times N$(data bit width of a path), which becomes very large for larger N. However, if the given sequences are not apparently similar, we do not need the alignment. Therefore, in our approach, two types of circuits are configured on FPGA[15, 17]. With the first type circuits, the similarity among sequences are checked by computing only the scores. Then, the second type circuits are configured on the FPGA, and the alignments are calculated for the sequences with high similarity (score) by storing all causal connections. In the following discussion, we focus on the first type circuits.

In our approach, N-dimensional dynamic programming is achieved by repeating two-dimensional dynamic programming along other dimensions in order to reduce the size of the score matrices which have to be cached on the



(A) Parallel Processing of two dimensional dynamic programming

(B) Parallel Processing of three dimensional dynamic programming

Fig. 3. Parallelism in Dynamic Programming

FPGA (for the protein sequences (k=24), the total size of the score matrix becomes 324K words when N=4). Suppose that we repeat the following procedure for four-dimensional dynamic programming (a four-dimensional score matrix $\mathrm{M_S}[a[x], b[y], c[z], d[t]]$ is used).

1. Calculate the alignment between two sequences (a and b) without changing other two sequences ($c[z] = C_k$ and $d[t] = D_l$; C_k and D_l are constants).
2. Increment z, and then t ($c[z]$ or(and) $d[t]$ is changed).

Then, we need only a part of the four-dimensional matrix, which is a two-dimensional score matrix ($\mathrm{M_S}[a[x], b[y], C_k, D_l]$) in the first step of the procedure. However, we need different two-dimensional score matrix when the value of $c[z]$ or $d[t]$ is changed. In our implementation, two-dimensional score matrices are implemented using dual-port RAMs in FPGA, and score matrices for next $b[z]$ or/and $d[t]$ (namely next parts of the four-dimensional score matrix) are downloaded from external RAMs on the FPGA board in parallel with the computation of scores. The number of score matrices which are download during the computation becomes 2^{N-2}. Thus, with a certain value of N, the downloading time of the next score matrices exceeds the time of the computation of the two-dimensional dynamic programming, and becomes the bottleneck of this approach.

In the following discussion, suppose that X, Y, Z and T are length of sequences placed along x, y, z and t axes, and W_x, W_y, W_z and W_t are part of sequences which can be processed continuously without extra input/output for boundary data. Figure 4 shows how three-dimensional dynamic programming is executed by the repetition of the two-dimensional dynamic programming. In Figure 4, processing of $W_x \times W_y$ nodes (two-dimensional dynamic programming) in the *scan window* (gray square in the figure) is scanned along z axis (the black arrow shows the *scan line*). When the scan window reaches at the end of z axis, it is shifted along y axis by W_y, and is scanned along z axis again. After processing $W_x \times Y \times Z$ nodes, the scan window is shifted down along x axis by W_x, and the same procedure is repeated.

Figure 5 shows the data input/output for the three-dimensional dynamic programming. In Figure 5(A), two dark gray rectangles show the inputs to the scan window (light gray square), and two rectangles with slanted lines show the output by the scan window. The outputs are stored, and used for the computation of other scan lines. In Figure 5(B), in order to calculate scores in the current

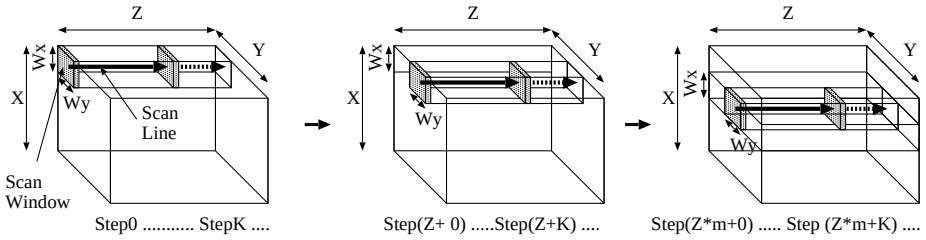


Fig. 4. Three-Dimensional Dynamic Programming

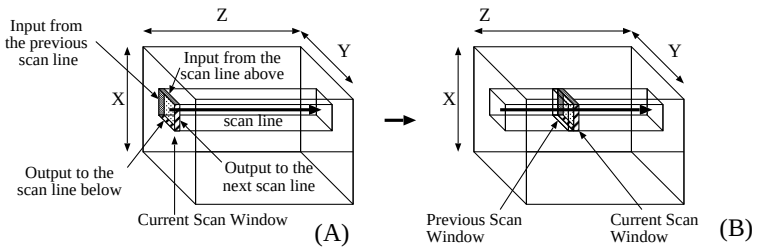


Fig. 5. Boundary Data for Three-Dimensional Dynamic Programming

scan window, data in previous scan window are also necessary (those data are not necessary in Figure 5(A), because the scan window is placed at the boundary on the search space, and boundary conditions are given instead of those data). Therefore, the data in previous scan window are held on FPGA.

Figure 6 shows the *scan cube* for four-dimensional dynamic programming (a cube is used instead of the window). Processing of nodes in the cube (size is $W_x \times W_y \times W_z$) is scanned along t axis, changing positions of the scan line. In order to calculate scores of the nodes in the cube, the scan window in the cube (light gray square in Figure 6(A)) is scanned along z axis. Suppose that current cube is on $(x, y, z, t{=}\mathrm{C_k})$. In order to start the calculation of the scan window (Figure 6(B)(1)), we need scores in dark gray parts and scores in the previous cube along t axis ($(x, y, z, t{=}\mathrm{C_k}\text{-}1)$ which are temporally held on the FPGA (not shown in the figure) as boundary data. Among these data, two dark gray rectangles in the figure can be obtained while calculating the scores of the nodes in the scan window. However, data in the dark gray square (the last scan window in the previous scan cube along z axis) need to be loaded before starting the calculation, because the size of data is large, and can not be loaded in parallel with the computation. The outputs by the scan window are two rectangles with slanted lines. When the scan window is in the cube (figure 6(B)(2)), scores calculated in the previous scan window are held on the FPGA, and used for the calculation of the current scan window (the scores in the previous cube along t axis which are held on the FPGA are also used). When the scan window reaches at the end of the cube, scores in the current window are stored for later processing (figure 6(B)(3)). In this processing of the scan cube, there are two types of data;

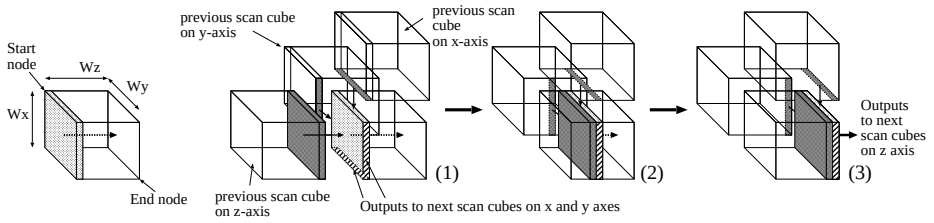


Fig. 6. Four-Dimensional Dynamic Programming

1. data which can be loaded, and output in parallel with the computation of the scores of the nodes in the scan window (two dark rectangles in Figure 6(B)(1,2,3)), and
2. data which have to be loaded before the computation (dark gray square in Figure 6(B)(1)) and which have to be stored after the computation (dark gray square in Figure 6(B)(3)).

The total clock cycles by our approach can be estimated as follows, when the data width of each element in score matrices is 16 bits, and the external memory banks run at the same speed as the circuit on the FPGA. In the following equations, the first term chooses the maximum of the computation time of the scan window ($W_x + W_y$) and the time to update score matrices which is executed in parallel with the computation. In other terms, constant values show the time to download score matrices, and other values show the time to input/output boundary data (some matrices can not be loaded in parallel with the computation, and we need to download them when $c[z], d[t]$ and so on are changed).

Three-Dimensional:

$$\max\left\{\begin{array}{c} W_x+W_y \\ 24^2/4 \end{array}\right\} \times \frac{XYZ}{W_xW_y}$$

Four-Dimensional:

$$\max\left\{\begin{array}{c} W_x+W_y \\ 24^2/8\times 2 \end{array}\right\} \times \frac{XYZT}{W_xW_y} + \max\left\{\begin{array}{c} 24^2/2 \\ W_xW_y\times 2/5 \end{array}\right\} \times \frac{XYZT}{W_xW_yW_z}$$

Five-Dimensional:

$$\max\left\{\begin{array}{c} W_x+W_y \\ 24^2/16\times 4 \end{array}\right\} \times \frac{XYZTU}{W_xW_y} + \max\left\{\begin{array}{c} 24^2/4\times 2 \\ W_xW_y\times 2/5 \end{array}\right\} \times \frac{XYZTU}{W_xW_yW_z} + \max\left\{\begin{array}{c} 24^2/2 \\ W_xW_yW_z\times 2/5 \end{array}\right\} \times \frac{XYZTU}{W_xW_yW_zW_t}$$

Six-Dimensional:

$$\max\left\{\begin{array}{c} W_x+W_y \\ 24^2/32\times 8 \end{array}\right\} \times \frac{XYZTUV}{W_xW_y} + \max\left\{\begin{array}{c} 24^2/5\times 4 \\ W_xW_y\times 2/5 \end{array}\right\} \times \frac{XYZTUV}{W_xW_yW_z} + \max\left\{\begin{array}{c} 24^2/5\times 8 \\ W_xW_yW_z\times 2/5 \end{array}\right\} \times \frac{XYZTUV}{W_xW_yW_zW_t} + \max\left\{\begin{array}{c} 24^2/5\times 7 \\ W_xW_yW_zW_t\times 2/5 \end{array}\right\} \times \frac{XYZTUV}{W_xW_yW_zW_tW_u}$$

In the equations above, $\{W_x, W_y, W_z, W_t, W_u\}$ are parameters which decide the performance, and have to be chosen so that the maximum performance can be realized under given hardware resources (the size of the FPGA, and the memory bandwitdh). For example, in our current implementation on ADM-XRC-II (FPGA board byits Alpha Data) with one Xilinx XC2V6000, $\{W_x, W_y, W_z, W_t\}$ are $\{10,64,6,3\}$ for five-dimensional dynamic programming, and it takes about 1.35×10^4 seconds to calculate the alignment, when the length of the sequences is 256. This performance is more than 100 times of Pentium 4 2GHz[17], but is still too slow for comparing longer sequences.

4 Multidimensional Dynamic Programming on a Distributed System

Figure 7(A) shows the search space in three-dimensional dynamic programming. With one FPGA, the computation of the scan window is started from the left-hand side of *box 11*, and the scan window is scanned along z axis (*scan line*). After finishing *box 11*, the scan window moves to *box 12*, and the computation of

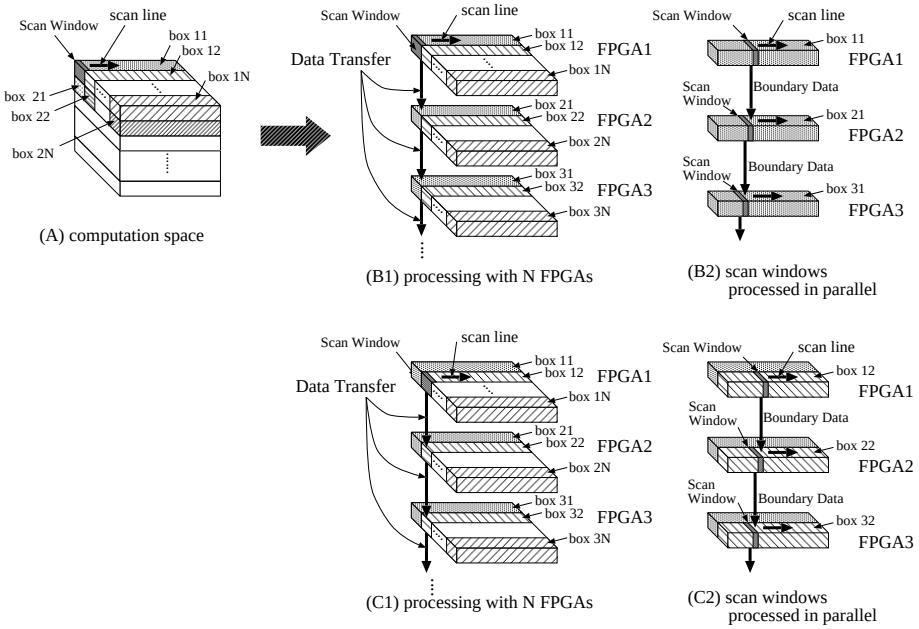


Fig. 7. Parallel Processing with Multiple FPGAs

the scan window is repeated. Figure 7(B1) shows how to divide the search space. In Figure 7(B1), FPGA_k processes *box k1 - kN* sequentially. When the first scan window in *box 11* is processed by FPGA1, the boundary data on its bottom are transferred to FPGA2. Then, FPGA2 starts the computation of the first scan window in *box 21*. In the same way, FPGA3 starts the computation of the first scan window in *box 31* as soon as the boundary data for the scan window arrive from FPGA2. Figure 7(B2) shows only the boxes which are processed in parallel. In this parallel processing, data transfer can be overlapped with the computation of scan windows. After finishing the computation of *box 11*, FPGA1 starts the computation of *box 12*, and FPGA2 also starts the computation of *box 22* (Figure 7(C1)(C2)).

Figure 8 shows when the computation of the scan window can be started on FPGA1 and FPGA2. The gray boxes in Figure 8 shows the first term of the equation in Section 3. During the computation of a scan window in FPGA1, its boundary data are sent to FPGA2, and FPGA2 starts the computation of its scan window using the boundary data. Figure 8(A) shows the flow of the computation when the data transfer is faster than the computation of the scan window, and Figure 8(B) shows the flow when it is slower. In Figure 8(B), each FPGA becomes idle to wait for sending its boundary data to its successor, and for the arrival of the boundary data from its predecessor.

Data transfer delay is not important in our computation method. The reason is as follows. FPGA1 can continue its computation until it finishes all the computation assigned to FPGA1, and the data transfer can be overlapped with the computation of the scan windows. FPGA2 becomes idle when waiting for the

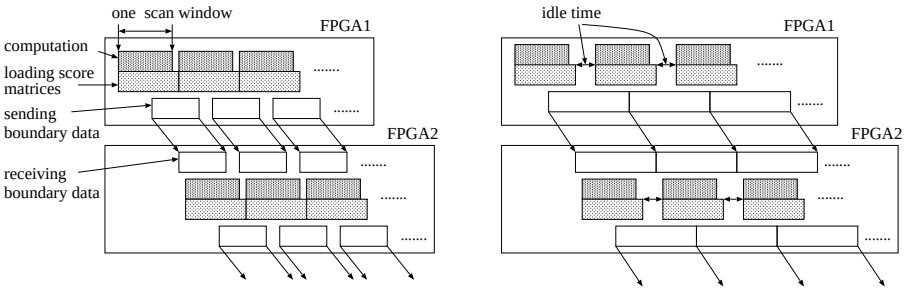


Fig. 8. Flow of the computation on FPGA1 and FPGA2

first arrival of the boundary data because of the data transfer delay, but after that, FPGA2 can continue its computation as far as the boundary data arrive within a certain delay. Therefore, the increase of the computation time by the data transfer delay is only

the data transfer delay × (the number of FPGAs - 1)

in the total computation time.

Figure 9 shows a distributed system for our computation method. In our approach, the search space is divided along x axis as shown in Figure 7(B1). When the number of FPGAs(N) is smaller than the number of the divided search spaces, some FPGAs have to process several of them sequentially (for example, FPGA$_i$ processes the *i-th*, *(N+i)-th*, *(2N+i)-th* spaces, and so on). Therefore, the nodes are connected as a ring. Each node on the system consists of an FPGA board with one FPGA, its host processor, and two network interface cards. With two network interface cards, each node receives boundary data from its predecessor, and send new boundary data to its successor.

5 Estimated Performance

We have implemented two circuits (four-dimensional and five-dimensional homology search) on XC2V6000, and they run at 36.6MHz and 31.0MHz respectively. The main reason of the low operational frequency is selectors to choose the maximum $2^N - 1$ candidates.

We are now testing the computation method using two nodes (two FPGA boards and their host processors) connected by Ethernet (100Mbps). Figure 10 shows the performance of the computation method which is estimated based on our experiments (five-dimensional, and legth of all sequences is 256). Boxes

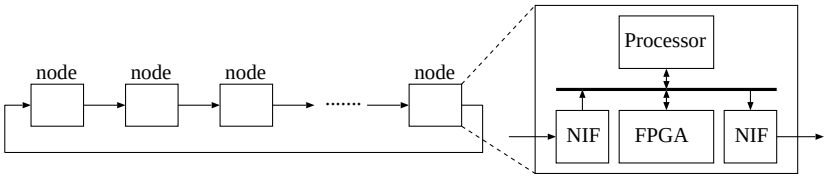


Fig. 9. A distributed system

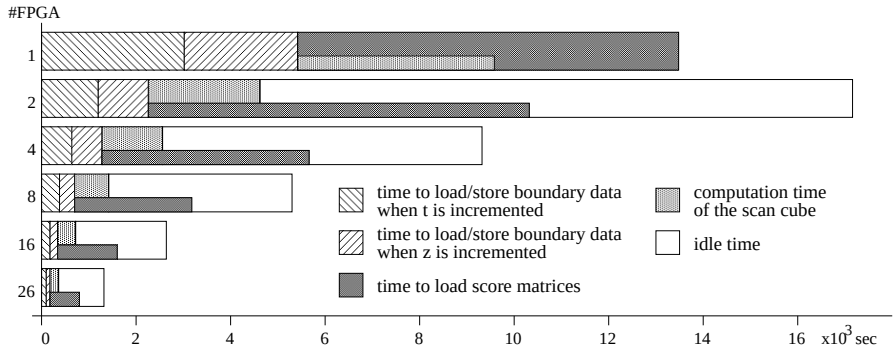


Fig. 10. Estimated performance on the distributed system

with slanted lines correspond to the second and the third terms of the equation shown in Section 3, and grey boxes correspond to the first term (the computation time, and the downloading time of the score matrices which can be executed in parallel with the computation). The size of the scan cube is {10,64,6,3} for non-distributed processing by one FPGA, and {10,32,14,3} for the distributed processing by more than one FPGA. These sizes are dicided so that the maximum performance can be achieved in each case. In the five-dimensional dynamic programming, the time to download score matrices is larger than the computation time. Therefore, we need to minimize the downloading time when processing by one FPGA. However, the downloading time can be hidden by the idle time caused by the slow data transfer on the distributed system, which allows us to focus to minimize the computation time. Because of the lack of the throughput for data transfer, the idle time occupies more than half of the total computation time when the number of FPGAs is larger than one. The computation time with two FPGAs is larger than one FPGA. However, we can obtain performance gain as the number of FPGAs increases. The performance gain becomes 5.1 times with 16 FPGAs, and about 10 times with 26 FPGAs. With 26 FPGAs, each FPGA processes only one divided search space, because the search space is divided to 26 sub-spaces ($X/W_x = 256/10$).

The data transfer delay is not important in our computation method as described in Section 4, when the computation time by each FPGA is large enough. When the number of FPGAs is N, the increase of the total computation time is abount $N \times d$ seconds if the data transfer daley becomes d second longer. This increase is very small compared with the total computation time.

6 Conclusions and Future Works

In this paper, we described a computation method for the multidimensional dynamic programming on distributed systems. The method is now being tested using two nodes connected by Ethernet. The data transfer speed of Ethernet (100 Mbps) is not enough, but according to our experiments, it is possible to achieve 5.1 times speedup with 16 nodes. The performance is affected only a little by the data transfer delay when comparing long sequences. Therefore, our method can be mapped on any kinds of networks with large delays.

We still have two major works. First, we need to evaluate the method using more FPGA boards, and then using more FPGA boards placed at distant places. Second, the size of boundary data can be compressed less than half, because two continuous data on the boundary have same values with high probability. We need to implement circuits to compress and uncompress the boundary data on FPGAs.

References

1. National Center for Biotechnology Information (NCBI), "NCBI-GenBank Flat File Release 137.0", *http://www.ncbi.nlm.nih.gov/*, Aug 2003.
2. European Molecular Biology Laboratory (EMBL), *http://www.ebi.ac.uk/embl/*.
3. DNA Data Bank of Japan (DDBJ), *http://www.ddbj.nig.ac.jp/*.
4. Henikoff, S. and Henikoff, J.G.: "Amino Acid Substitution Matrices from Protein Block", Proc. Natl. Acad. Sci. 89, pp.10915-10919, 1992.
5. Jones, D. T. et. al: "The Rapid Generation of Mutation DataMatrices from Proten Sequences", CABIOS 8, pp.275-282, 1992.
6. Stephen F. Altschula, Warren Gisha, Webb Millerb, Eugene W. Meyersc, and David J. Lipman, "Basic Local Alignment Search Tool", *Journal of Molecular Biology*, Vol.215, Issue 3, pp.403-410, 1990.
7. Stephen F. et al, "Gapped BLAST and PSI-BLAST: a new generation of protein database search programs", *Nucleic Acids Research*, Vol.25, No.17, pp.3389-3402, 1997.
8. William R. Pearson and David J. Lipman "Improved tools for biological sequence comparison", *Proceedings of the National Academy of Sciences of the USA*, Vol.85, pp.2444-2448, 1988.
9. PARACEL, "GeneMatcher2", *http://www.paracel.com/*.
10. Dominique Lavenier, "SAMBA Systolic Accelerators For Molecular Biological Applications", *Technical Report RR-2845*, 1996.
11. C. Thomas White, et al, "BioSCAN: A VLSI-Based System for Biosequence Analysis", *IEEE International Conference on Computer Design: VLSI in Computer & Processors*, Vol.147, pp.504-509, (1991).
12. TimeLogic Corporation, "Decypher bioinformaticsacceleration solution", *http://www.timelogic.com/products.html*, 2002.
13. Kiran Puttegowda, William Worek, Nicholas Pappas, Anusha Dandapani, and Peter Athanas, "A Run-Time Reconfigurable System for Gene-Sequence Searching", *International VLSI Design Conference*, pp.(to appear), 2003.
14. Steven A. Guccione and Eric Keller, "Gene Matching Using JBits", *International Conferenece on Field-Programmable Logic and Applications*, pp.1168-1171, 2002.
15. Yoshiki Yamaguchi, Yosuke Miyajima, Tsutomu Maruyama, Akihiko Konagaya, "High Speed Homology Search with Run-time Reconfiguration", *International Conferenece on Field-Programmable Logic and Applications*, pp.281-291, 2002.
16. Yoshiki Yamaguchi, Tsutomu Maruyama, Akihiko Konagaya, "Three-Dimensional Dynamic Programming for Homology Search", *International Conferenece on Field-Programmable Logic and Applications*, pp.505-514, 2004.
17. S. Masuno, T. Maruyama, Y. Yoshiki and A. Konagaya, "Multidimensional Dynamic Programming for Homology Search", *International Conferenece on Field-Programmable Logic and Applications*, pp.173-178, 2005.