The review of the acceleration of Smith-Waterman algorithm by using CUDA-enable GPU

Chenyang Jing[1, a], Le Zhang[1,b*]

[1]College of Computer and Information

Science, Southwest University,

Chongqing 400715, China

[a]longnigh@email.swu.edu.cn, [b]zhanglcq@swu.edu.cn

Keywords: Acceleration, Sequence alignment, Smith-Waterman algorithm, CUDA.

Abstract. Smith-Waterman (SW) algorithm, which calculates the similarity between two given sequences, is broadly used in bioinformatics research field. However, the time complexity of the SW algorithm prevents it from being used for long sequence alignment. Since SW algorithm is based on dynamic programing, using single instruction multiple data parallel computing algorithm can significantly reduce the computing cost. For this reason, this review introduces three commonly used parallel computing algorithms based on Compute Unified Device Architecture (CUDA) for SW algorithm acceleration as well as illustrates their advantages and disadvantages.

Introduction

Sequence alignment and comparison are broadly used in bioinformatics and molecular biology. It is a way of arranging the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences [1]. Gaps are inserted between the residues so that same or mismatch characters are aligned in successive columns. When align a query sequence to all the sequences in a database, the aligner computes a score which represents the degree of similarity between the two sequences.

Sequence alignment generally falls into two categories: global and local alignment. They both employed dynamic programming [4] algorithm. Global alignment was proposed by Needleman and Wunsch in 1970 [2]. Local alignment was designed by Smith and Waterman that identifies regions of similarity within two given sequences regardless of how much imparity between them [3]. Local alignments are more useful for the dissimilar sequences that are suspected to contain regions of similarity or similar sequence motifs within their larger sequence context. Smith-Waterman algorithm is the evolution of Needleman-Wunsch algorithm by setting zero as the minimum score to prevent calculated negative similarity. After that, Gotoh [5] improved Smith–Waterman algorithm(SW) by introducing linear gap-penalty function.

Duo to the high computing cost of SW algorithm, heuristic algorithms were proposed to solve the problem by executing the local alignment via generating seeds and extending them. For example, FASTA [6], proposed by Arson and Lipman in 1985 and improved in 1988 [7], and BLAST, proposed by Altschul et al., [8, 9] in 1990 are two famous heuristic algorithms. However, heuristic algorithms have to sacrifice the computing accuracy [10] to gain faster computing speed than SW algorithm.

To obtain both the computing accuracy and speed, serveral special-purpose hardware implementations have been developed [11], such as field-programmable gate arrays (FPGAs) or super-computers, Paracel’s CeneMatcher, Compugen’s Bioccelerator and TimeLogic’s DeCypher [10]. Although these special-purpose hardware can offer high performance parallel alignment technique, they are too expensive for common users to use.

Compute Unified Device Architecture (CUDA) developed by NVIDIA company can efficiently work for high performance computing in scientific Computing [12], computational geometry [13], bioinformatics [14, 15], petroleum exploration and so on [16]. As a single instruction multiple data (SIMD) technology [10, 11, 17], it is good at speeding up SW algorithm.

This review focuses on how to employ CUDA technology to accelerate Smith–Waterman algorithm. We will introduce three commonly used methods for SW algorithm parallelization.

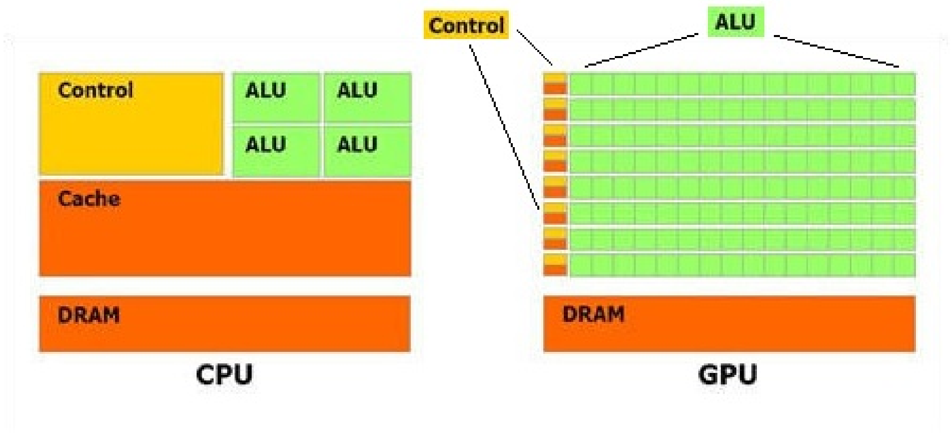


Figure 1: Different design philosophies for CPUs and GPUs [19]

Compute Unified Device Architecture (CUDA)

Graphic programming Unit (GPUs) performance's growth rate is faster than Moore's law as it applies to CPUs [16], and now, GPU becomes an alternative for high performance computing. GPGPU is a technique of using a GPU to perform computation in applications traditionally handled by the CPU [18]. NVIDIA and AMD, two major GPU manufacturers, both publish their GPGPU platforms, CUDA [16, 19] and CTM [20] respectively. CTM is based on assembly language which is not easy to handle, so CUDA is more popular than CTM.

CUDA-enable GPUs integrate a set of multiprocessors. Each multiprocessor contains several ALUs (Arithmetic Logic Unit), random access memory, cache and control unit (**Fig. 1**). CUDA makes it possible to directly access their graphic card and efficiently use massive threads. The GPU or graphic card seems as an independent device which used to execute the kernel. CUDA platform uses CUDA C/C++, a minimal set of extension of the C/C++ programming language and a runtime library [21].

CUDA program consists of two parts, the host code which executes on CPU and the kernel which executes on GPU. Usually, parallel codes are written in kernel function that can be run in every thread simultaneously. These threads are organized in a hierarchy consisting of so-called blocks and grids (**Fig. 2**). Each thread has its own unique ID (threadIdx, blockIdx). The threadIdx indicates the ID of the thread in a block and the blockIdx indicates the ID of the block in the grid. CUDA provides each thread access through its index pair threadIdx and blockIdx. The kernel call is similar to C language function call except it must explicitly specify the total size of a grid and block in a couple of triple angle brackets "<<<" and ">>>" as shown below.

kernel<<<size of grid, size of block, other configurations>>>(parameter list);

The CUDA programming model also assumes that both the kernel and the host code maintain their own separate memory, called device memory and host memory, respectively. CUDA program can employ different kinds of device memory. In order to write efficient CUDA programs, it is necessary for us to understand the different characteristics and performances among different memory types in CUDA.

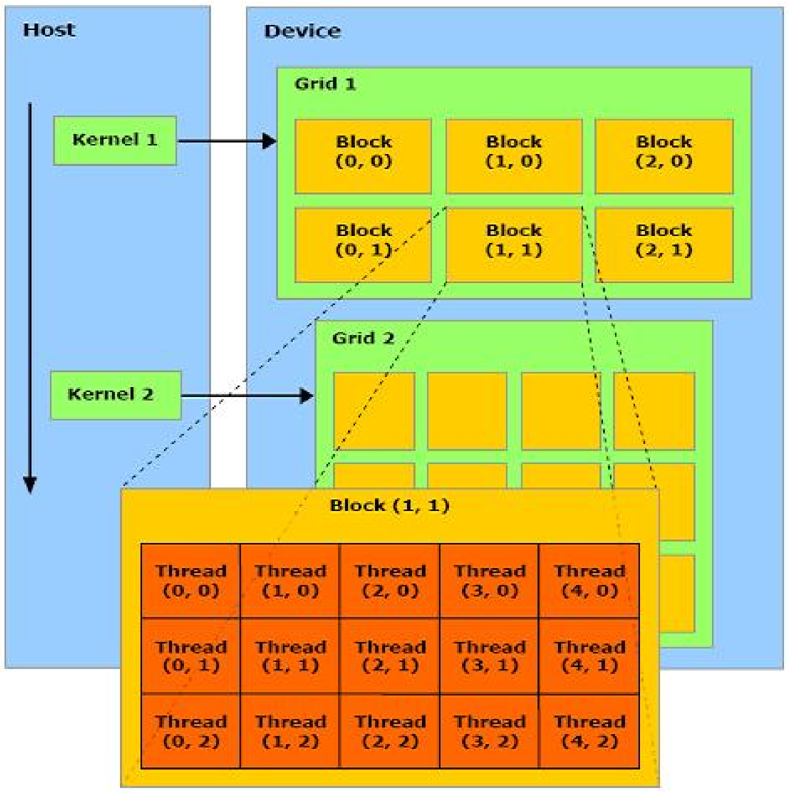


Figure 2: CUDA thread organization [19]

CUDA-enable GPU accelerating local alignment algorithm

As we mentioned before, local alignment is more accurate than global alignment, while it is not capable of processing long sequence alignment limited to its computing capacity. One of the most wildly used methods for local alignment is Smith-Waterman algorithm. Smith–Waterman algorithm(SW) employs scoring matrix to evaluate the similarity between two sequences [22] as following.

Consider two strings S_1 and S_2 of length l_1 and l_2. To identify the common subsequences, the Smith-Waterman algorithm computes the similarity $H_A(i,j)$ of the two sequences, ending at position i and j of the two sequences S_1 and S_2. The computation of $H_A(i,j)$, for $1 \le i \le l_1$, $1 \le j \le l_2$, is given by the following recurrences:

$$H_A(i,j) = \max\{0, E(i,j), F(i,j), H_A(i-1,j-1) + sbt(S_1[i], S_2[j])\},$$

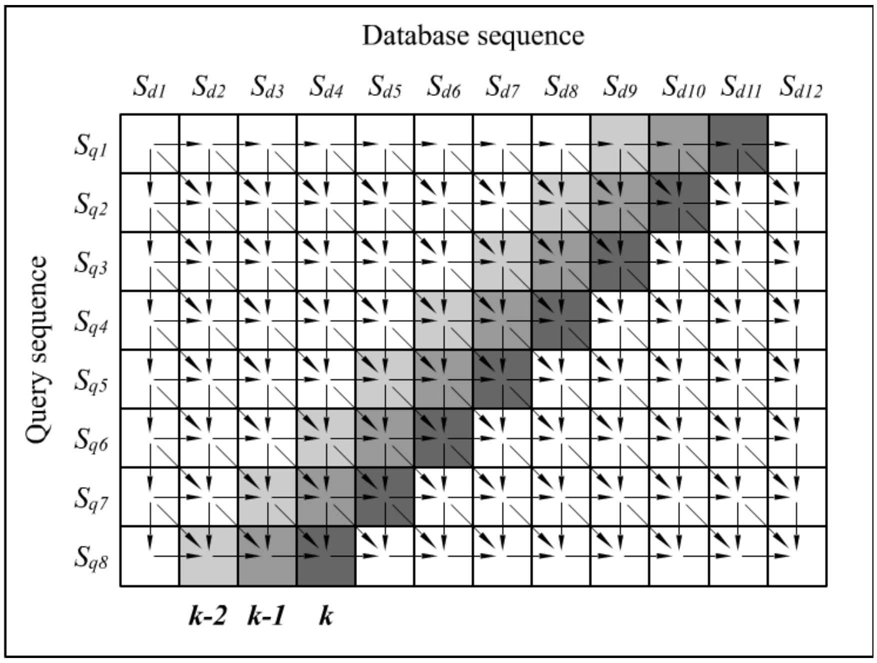
$$E(i,j) = \max\{H_A(i,j-1) - \alpha, E(i,j-1) - \beta\},$$

$$F(i,j) = \max\{H_A(i-1,j) - \alpha, F(i-1,j) - \beta\}. \quad (1)$$

where sbt is a character substitution cost table. The initialization of these values is given $H_A(i,0) = E(i,0) = H_A(0,j) = F(0,j) = 0$ for $0 \le i \le l_1$, $0 \le j \le l_2$. Multiple gap costs are taken into account as follows: α is the cost of the first gap; β is the cost of the following gaps. This type of gap cost is known as affine gap penalty. Some applications also use a linear gap penalty, that is, α=β. For linear gap penalties, the above recurrence relations can be simplified as Eq. 2

$$H_L(i,j) = \max\{0, H_L(i,j-1) - \alpha, H_L(i-1,j) - \alpha, H_L(i-1,j-1) + sbt(S_1[i], S_2[j])\}. \quad (2)$$

where $H_L(i,j)$ represents a similarity value. The two segments of S_1 and S_2 producing this value can be determined by a traceback procedure [23].


Database sequence
S_{d1} S_{d2} S_{d3} S_{d4} S_{d5} S_{d6} S_{d7} S_{d8} S_{d9} S_{d10} S_{d11} S_{d12}
Query sequence
S_{q1} S_{q2} S_{q3} S_{q4} S_{q5} S_{q6} S_{q7} S_{q8}
k-2 k-1 k
Figure 3: SW data dependencies

We must consider that, the order of the computation of the value in each cell of matrix is strict. The value of any cell must be computed after the value of all cells to the left and above it. Fig. 3 shows the data dependencies in calculation of the scoring matrix.

Since the time complexity of Smith-Waterman algorithm is O (mn), it is too slow to process long sequence alignment. In the following, we will describe several parallel computing algorithms to accelerate Smith-Waterman algorithm.

Coarse-grain Accelerator. This type of method utilizes one independent thread of the GPU to align one query sequence with one database sequence. For each thread, the DP matrix is divided into a group of vectors. Every vector is parallel to the query sequence, and the computation of the DP matrix is processed vector by vector.

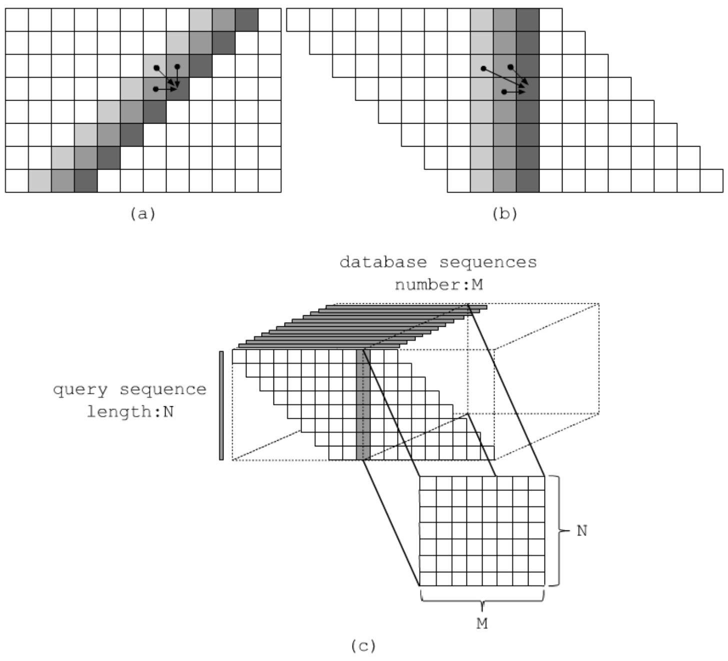


Figure 4: Fine grain method [23].

Since there will be a large time difference for the various thread to process a short or long sequence, it will result in the long waiting time for the different threads. For this reason, Manavski et al., [30], pre-ordered the database sequences by the length of them and grouped them with the almost same length. And then, they processed these groups with almost same length of sequences one by one by parallel computing algorithm. Manavski et al., [30] implemented this pre-order strategy and its performance reaches 1.8 giga cell updates per second (GCUPS). Then, Liu et al., [24-26] and Rognes et al., [10], implemented this method in the CUDA and microprocessors, respectively.

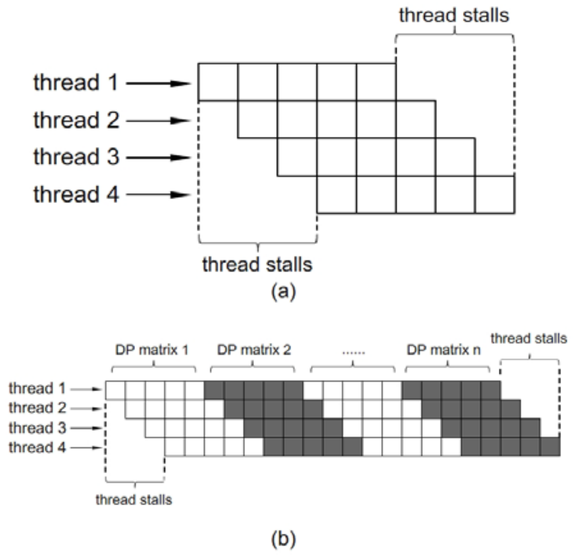


Figure 5: Multiple DP matrices connection to amortize the time cost [27]

Fine-grain Accelerator. Fine-grain method benefits the advantage of multithread to compute one DP matrix in parallel by eliminating the data dependencies. Fig. 3 shows data independencies between the cells in the same anti-diagonal of the DP matrix. By exploiting the parallel characteristic of the anti-diagonal, each thread computes a cell in the anti- diagonal and it needs L_q (the length of query sequence) thread in total.

As is shown by Fig. 4a, the cells in one anti-diagonal are shifted in a column. All the cells in one anti-diagonal can be computed in each iteration (see Fig. 4b). The data dependencies between anti-diagonals are shifted into each columns. As shown in Fig. 4c, computing one DP matrix of query sequence with length N needs N thread. Utilizing 2-dimension character of CUDA multithreading, M database sequences can be processed simultaneously when employing M×N threads.

This rectangle transformation can utilize the multithreading to speed up alignment. However, a few threads will still stall when processing non-major anti-diagonal. To solve this problem, Yang Liu et al., [27] innovated an algorithm for multiple DP matrices connection to amortize the time cost of stalls as is shown by Fig. 5.

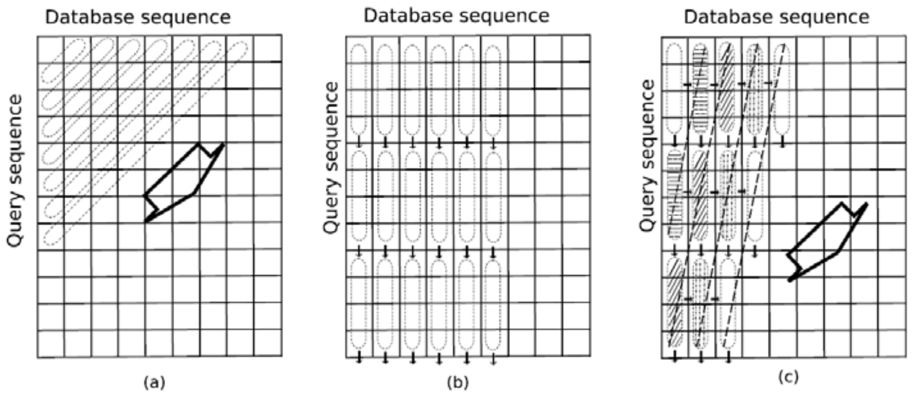


Figure 6: Parallel alternatives to perform the SW [28]

Also, Sanchez et al., [28], combines the fine-grain accelerator and coarse-grain accelerator. As shown in Fig. 6, this mechanism processes the vector in parallel to the query sequence. It processes the vector from the next column in anti-diagonal direction at the same time. Sanchez et al., [28] claimed that the implementation obtains a 30% reduction in the execution time relative to the Coarse-grain [10, 24-27, 30] and Fine-grain [23-29, 31] methods.

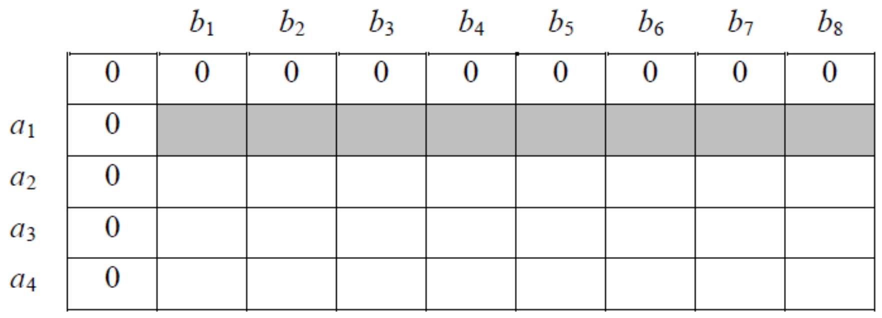


Figure 7: Compute one row of the DP matrix in an iteration [29]

Redefine Recursive Formula Method. When we consider the gap open and gap extension as the same cost, the formula will become:

$$h_{i,j} = \max\left\{h_{i-1,j-1} + S\left[S_q[i], S_d[j]\right], h_{i-1,j} - p, h_{i,j-1} - p, 0\right\}. \tag{3}$$

In order to compute one row of the DP matrix in an iteration (see Fig. 7), it must overcome the data dependencies in a row.

define:

$$h'_{i,j} = \max\left\{h_{i-1,j-1} + S\left[S_q[i], S_d[j]\right], h_{i-1,j} - p, 0\right\}. \tag{4}$$

then,

$$h_{i,j} = \max\{h'_{i,j}, h_{i,j-1} - p, 0\}. \tag{5}$$

Use h' to replace h:

$$h_{i,j} = \max\{h'_{i,j}, h'_{i,j-1} - p, h_{i,j-2} - 2p, 0\}. \tag{6}$$

Finally, we can obtain the final formula:

$$h_{i,j} = \max\{h'_{i,j}, h'_{i,j-1} - p, h'_{i,j-2} - 2p, \dots, h'_{i,1} - (j-1)p, h'_{i,0} - jp, 0\}. \tag{7}$$

It is clear that all values in ith row are dependent on the values of $h'_{i,j}$, and all values of $h'_{i,j}$ are dependent on the values $h_{i-1,j}$, so there is no data dependency in the same row and the values of h' can be computed in advance. Yeim-Kuan Chang et al., [29] implemented this method and it is 2-4 times faster than other methods.

By using this redefined formula, computing a DP matrix with m rows and n columns just needs n threads and m time passes. However, the computing time of one row depends on the last cell. Chang's method employs prefix max scan method [29] to reduce the computation complexity for each row.

Given the input array [a_1, …, a_n], the prefix max scan method generate the output array [M(a_1, a_1), M(a_1, a_2), …, M(a_1, a_n)], where M(a_1, a_j) is the maximal value from a_1 to a_j. Fig. 8 shows the mechanism of the prefix max scan method. Assume n = 8, the input is the array of $h'_{i,1} + (n-1) \times g$, $h'_{i,2} + (n-2) \times g$, …, $h'_{i,n}$. The output will be an array of $h'_{i,1} + (n-1) \times g$,

$M(h'_{i,1}+(n-1)\times g, h'_{i,2}+(n-2)\times g)$, …, $M(h'_{i,1}+(n-1)\times g, h'_{i,n})$. After adding output array and the array of $-(n-1)\times g, -(n-2)\times g, \ldots, -(n-n)\times g$ together, the result in ith row is obtained. The computation time for each row is O(log2n). The total computation time for the matrix is m×O(log2n) = O(mlog2n).

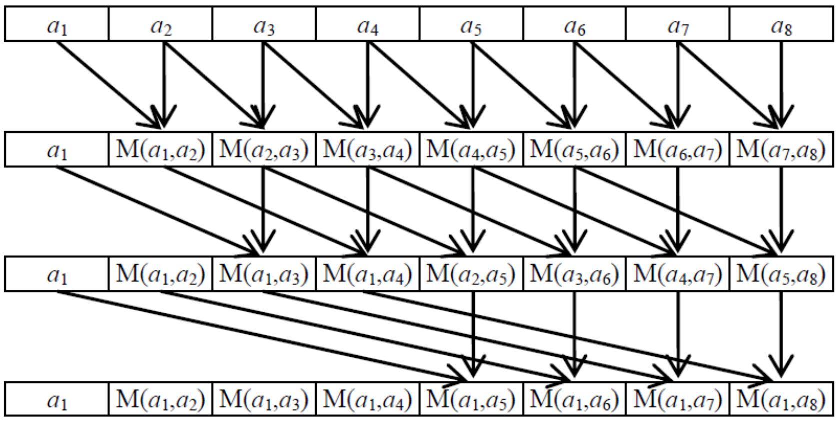

a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8
a_1 M(a_1,a_2) M(a_2,a_3) M(a_3,a_4) M(a_4,a_5) M(a_5,a_6) M(a_6,a_7) M(a_7,a_8)
a_1 M(a_1,a_2) M(a_1,a_3) M(a_1,a_4) M(a_2,a_5) M(a_3,a_6) M(a_4,a_7) M(a_5,a_8)
a_1 M(a_1,a_2) M(a_1,a_3) M(a_1,a_4) M(a_1,a_5) M(a_1,a_6) M(a_1,a_7) M(a_1,a_8)

Figure 8: Prefix max scan [29]

Query-profile. Query-profile is an auxiliary method for improving the performance of the parallel acceleration algorithm [10, 24-27, 30]. For example, Query-profile method was employed by Rognes et al., [10], Manavski et al., [30] and Liu et al., [24-26] for coarse grain acceleration algorithm.

When the same query sequence is compared with many different database sequences, a simple speed improvement is achieved by creating a kind of score profile for the query sequence. This profile, which can be considered as a query-specific substitution score matrix, is computed only once for the entire search. Instead of indexing the original substitution-score matrix by the query-sequence symbol and the database-sequence symbol, the new matrix is indexed by the query sequence position and the database sequence symbol. The score for matching symbol A (for alanine) in the database sequence with each of the symbols in the query sequence is stored sequentially in the first matrix row, followed by the scores for matching symbol B (ambiguous) in the next row, and so on [10].

Summary

Smith-Waterman algorithm is a high-sensitivity and computationally-intensive sequence alignment algorithm that is fundamental to the analysis of proteins and genes. Since the high time complexity of SW algorithm, it is necessary to speed up the process of SW algorithm.

The coarse grained acceleration method utilizes different threads to process several DP matrixes simultaneously. So there is no data dependency in this method. The fine grained acceleration method takes advantage of anti-diagonal of the DP matrix to avoid data dependency among threads. Redefine recursive formula method redefines the recursive formula and eliminates the data dependency among the cells in a row of DP matrix.

The coarse grained acceleration method performance reaches a peak of 1.8 GCUPS or more [10, 24-27, 30]. The advantage of this method is that it processes several alignments at the same time. Since there will be a large time difference for the various thread to process a short or long sequence, the disadvantage of this method is that it will result in the long waiting time for the different threads.

The fine grained acceleration method performance reaches a peak of 0.7 GCUPS or more [23-29, 31]. The advantage of this method is that it utilizes multi-thread to compute a DP matrix to reduce computing time. The disadvantage is its huge time cost at the beginning and ending of the computation [23-29, 31].

The redefine recursive formula method redefines the recursive formula of Smith-Waterman algorithm and reduces the time complexity of Smith-Waterman algorithm from O(mn) to O(mlog2n) [29]. The advantage is to increase computing efficiency by reducing the algorithm complexity. The disadvantage is that it cannot give different penalty score for gap open and gap extension.

In general, there is no optimal parallel computing method to accelerate Smith-Waterman algorithm. In the recent future, we plan to integrate the advantages of these three algorithm to speedup the SW algorithm.

Acknowledgement

This work supported by the Natural Science Foundation of China under Grant No. 61372138, Chongqing excellent youth award and the Chinese Recruitment Program of Global Youth Experts.

References

[1] MOUNT D W. Sequence and genome analysis [J]. Bioinformatics: Cold Spring Harbour Laboratory Press: Cold Spring Harbour, 2004, 2

[2] NEEDLEMAN S B, WUNSCH C D. A general method applicable to the search for similarities in the amino acid sequence of two proteins [J]. J Mol Biol, 1970, 48(3): 443-53.

[3] SMITH T F, WATERMAN M S. Identification of Common Molecular Subsequences [J]. J Mol Biol, 1981, 147(1): 195-7.

[4] POLYANOVSKY V O, ROYTBERG M A, TUMANYAN V G. Comparative analysis of the quality of a global algorithm and a local algorithm for alignment of two sequences [J]. Algorithms Mol Biol, 2011, 6(1): 25.

[5] GOTOH O. An improved algorithm for matching biological sequences [J]. J Mol Biol, 1982, 162(3): 705-8.

[6] LIPMAN D J, PEARSON W R. Rapid and sensitive protein similarity searches [J]. Science, 1985, 227(4693): 1435-41.

[7] ALTSCHUL S F, LIPMAN D J. Trees, stars, and multiple biological sequence alignment [J]. SIAM Journal on Applied Mathematics, 1989, 49(1): 197-209.

[8] ALTSCHUL S F, GISH W, MILLER W, et al. Basic local alignment search tool [J]. J Mol Biol, 1990, 215(3): 403-10.

[9] ALTSCHUL S F, MADDEN T L, SCHAFFER A A, et al. Gapped BLAST and PSI-BLAST: a new generation of protein database search programs [J]. Nucleic Acids Res, 1997, 25(17): 3389-402.

[10] ROGNES T, SEEBERG E. Six-fold speed-up of Smith-Waterman sequence database searches using parallel processing on common microprocessors [J]. Bioinformatics, 2000, 16(8): 699-706.

[11] HUGHEY R. Parallel hardware for sequence comparison and alignment [J]. Computer applications in the biosciences: CABIOS, 1996, 12(6): 473-9.

[12] KR GER J, WESTERMANN R. Linear algebra operators for GPU implementation of numerical algorithms; proceedings of the ACM Transactions on Graphics (TOG), F, 2003 [C]. ACM.

[13] AGARWAL P K, KRISHNAN S, MUSTAFA N H, et al. Streaming geometric optimization using graphics hardware [M]. Algorithms-ESA 2003. Springer. 2003: 544-55.

[14] HORN D R, HOUSTON M, HANRAHAN P. Clawhmmer: A streaming hmmer-search implementatio; proceedings of the Proceedings of the 2005 ACM/IEEE conference on Supercomputing, F, 2005 [C]. IEEE Computer Society.

[15] CHARALAMBOUS M, TRANCOSO P, STAMATAKIS A. Initial experiences porting a bioinformatics application to a graphics processor [M]. Advances in Informatics. Springer. 2005: 415-25.

[16] nVidia: nVidia Compute Unified Device Architecture (CUDA) Programming Guide, version 1.0 [J]. 2007,

[17] SCHATZ M C, TRAPNELL C, DELCHER A L, et al. High-throughput sequence alignment using Graphics Processing Units [J]. BMC bioinformatics, 2007, 8(1): 474.

[18] FUNG J, TANG F, MANN S. Mediated reality using computer graphics hardware for computer vision; proceedings of the Wearable Computers, 2002(ISWC 2002) Proceedings Sixth International Symposium on, F, 2002 [C]. IEEE.

[19] NVidia CUDA [J].

[20] AMD CTM [J].

[21] NICKOLLS J, BUCK I, GARLAND M, et al. Scalable parallel programming with CUDA [J]. Queue, 2008, 6(2): 40-53.

[22] LAMBERT C, CAMPENHOUT J, DEBOLLE X, et al. Review of common sequence alignment methods: clues to enhance reliability [J]. Current Genomics, 2003, 4(2): 131-46.

[23] LIU W, SCHMIDT B, VOSS G, et al. Streaming algorithms for biological sequence alignment on GPUs [J]. Parallel and Distributed Systems, IEEE Transactions on, 2007, 18(9): 1270-81.

[24] LIU Y, MASKELL D L, SCHMIDT B. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units [J]. BMC research notes, 2009, 2(1): 73.

[25] LIU Y, SCHMIDT B, MASKELL D L. CUDASW++ 2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions [J]. BMC research notes, 2010, 3(1): 93.

[26] LIU Y, WIRAWAN A, SCHMIDT B. CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions [J]. BMC bioinformatics, 2013, 14(1): 117.

[27] LIU Y, HUANG W, JOHNSON J, et al. Gpu accelerated smith-waterman [M]. Computational Science–ICCS 2006. Springer. 2006: 188-95.

[28] S NCHEZ F, SALAM E, RAMIREZ A, et al. Parallel processing in biological sequence comparison using general purpose processors; proceedings of the Workload Characterization Symposium, 2005 Proceedings of the IEEE International, F, 2005 [C]. IEEE.

[29] CHANG Y-K, CHEN D-Y. Fast Sequence Alignment Method Using CUDA-enabled GPU [J].

[30] MANAVSKI S A, VALLE G. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment [J]. BMC bioinformatics, 2008, 9(Suppl 2): S10.

[31] LIU W, SCHMIDT B, VOSS G, et al. Bio-sequence database scanning on a GPU; proceedings of the Parallel and Distributed Processing Symposium, 2006 IPDPS 2006 20th International, F, 2006 [C]. IEEE.