Computational Biology and Chemistry 58 (2015) 62–68

Contents lists available at ScienceDirect

Computational Biology and Chemistry

journal homepage: www.elsevier.com/locate/compbiolchem

CUDA ClustalW: An efficient parallel algorithm for progressive multiple sequence alignment on Multi-GPUs

Che-Lun Hung [a], Yu-Shiang Lin [b], Chun-Yuan Lin [c,*], Yeh-Ching Chung [b], Yi-Fang Chung [c]

[a] Department of Computer Science and Communication Engineering, Providence University, 200, Sec. 7, Taiwan Boulevard, Shalu Dist., Taichung City 43301, Taiwan
[b] Department of Computer Science, National Tsing Hua University, 101, Sec. 2, Kuang-Fu Road, Hsinchu City 30013, Taiwan
[c] Department of Computer Science and Information Engineering, Chang Gung University, 259, Wen-Hwa 1st Road, Kwei-Shan, Taoyuan City 33302, Taiwan

ARTICLE INFO

Article history:
Received 13 November 2014
Received in revised form 14 May 2015
Accepted 14 May 2015
Available online 21 May 2015

Keywords:
Progressive multiple sequence alignment
GPU
CUDA
ClustalW
Parallel computing

ABSTRACT

For biological applications, sequence alignment is an important strategy to analyze DNA and protein sequences. Multiple sequence alignment is an essential methodology to study biological data, such as homology modeling, phylogenetic reconstruction and etc. However, multiple sequence alignment is a NP-hard problem. In the past decades, progressive approach has been proposed to successfully align multiple sequences by adopting iterative pairwise alignments. Due to rapid growth of the next generation sequencing technologies, a large number of sequences can be produced in a short period of time. When the problem instance is large, progressive alignment will be time consuming. Parallel computing is a suitable solution for such applications, and GPU is one of the important architectures for contemporary parallel computing researches. Therefore, we proposed a GPU version of ClustalW v2.0.11, called CUDA ClustalW v1.0, in this work. From the experiment results, it can be seen that the CUDA ClustalW v1.0 can achieve more than 33× speedups for overall execution time by comparing to ClustalW v2.0.11.

1. Introduction

In computational biology, sequence alignment is of priority concern and many methods have been developed to solve sequence alignment-related problems for biological applications. Needleman and Wunsch (Needleman and Wunsch, 1970) developed the well-known dynamic programming algorithm for solving global pairwise alignment problem. A similar algorithm was proposed by Smith and Waterman (Smith and Waterman, 1981) to solve the local pairwise alignment problem. Besides the pairwise alignment problem, the multiple sequence alignment problem was also elucidated and several methods were developed to obtain the optimal solution (Carrillo and Lipman, 1988). However, it was demonstrated that using the dynamic programming algorithm to solve the multiple sequence alignment problem is an NP-hard problem (Wang and Jiang, 1994). Many approximation and heuristic algorithms for multiple sequence alignment were developed in the past, i.e. ClustalW (Thompson et al., 1994), a progressive multiple sequence alignment tool. Progressive multiple sequence alignment (Thompson et al., 1994; Feng and Doolittle, 1987; Notredame et al., 2000a) is the commonly used approach to align a set of sequences by repeatedly aligning pairs of sequences and previously generated alignments. The idea of progressive multiple sequence alignment is to align pairs of sequences according to the orders in the phylogenetic tree (called guide tree) which was built by the similarity scores (formed as a distance matrix) calculated from each pair of sequences. Multiple sequence alignment tools have been used to discover DNA motif (Wong and Zhang, 2014), predict disease (Wong et al., 2013) and etc.

Due to the rapid growth of the biotechnology, such as next generation sequencing, the output size of sequencing data has increased at a rate that outpaces the Moore's law. In 2007, a single sequencing run could produce about one gigabase (Gb) size of sequence data. By 2011, it has approximately reached a terabase (Tb) size of sequence data in a single sequencing run. Over the past four years it has grown nearly 1000 times. When the problem instance is large, progressive alignment will be time consuming. Therefore, how to analyze a large number of sequences is an important issue. Parallel computing is a suitable solution to solve this problem. For example, ClustalW-MPI (Li, 2003) has been proposed to successfully solve the parallelization problem of distance matrix calculation by using message passing interface (MPI) library on a PC cluster. However, it needs a lot of budget to

* Corresponding author.
E-mail addresses: clhung@pu.edu.tw (C.-L. Hung), cyulin@mail.cgu.edu.tw (C.-Y. Lin).

http://dx.doi.org/10.1016/j.compbiolchem.2015.05.004
1476-9271/

maintain a PC cluster system. Current high-end graphics processing units (GPUs) are very popular in the high performance computing community due to contain up to hundreds cores per chip. GPU has massive multi-threaded processors; moreover, the thousands of threads can be executed simultaneously to fully utilize GPU computing power. Compute Unified Device Architecture (CUDA) (Nickolls et al., 2008) could access GPUs and has made the supercomputing available to the mass.

Several algorithms or tools have been ported on GPUs with CUDA in computational biology, such as MUMmerGPU (Schatz et al., 2007; Trapnell and Schatz, 2009), CUDA-MEME (Liu et al., 2010a), CUDA-BLASTP (Liu et al., 2011), and etc. For pairwise alignment, several works for Smith and Waterman algorithm have been implemented on GPUs (Manavski and Valle, 2008; Liu et al., 2009a, 2010b; Sandes and Melo, 2010). Most of them were based on the inter-task parallelization (Liu et al., 2009a) to calculate the similarity score (without alignment results) of each pair of input sequences by one thread. Only a few of literatures tried to use the intra-task parallelization (Liu et al., 2009a) to calculate the similarity score of each pair of input sequences by one thread block. Although the inter-task parallelization can achieve higher performance on GPUs than the intra-task parallelization, it is suitable for shorter sequences by comparing with the intra-task parallelization as mentioned in the literature (Liu et al., 2009a). The reason is that the sizes of device memory, registers and shared memory on GPU are limited. In the literature (Liu et al., 2009a), a threshold is set to 3072 for the length of database sequence. If the length of database sequence is less than the threshold, the Smith and Waterman computation is done by inter-task parallelization, otherwise by intra-task parallelization. Few works were presented to calculate the sequence alignment results not only similarity scores for long sequences by using Smith and Waterman algorithm on GPUs (Khajeh-Saeed et al., 2010; Sandes and Melo, 2011). Lee et al. (Lee et al., 2013) proposed a Smith-Waterman algorithm with a frequency-based filtration method on GPUs rather than merely accelerating the comparisons yet expending computational resources to handle such unnecessary comparisons. For multiple sequence alignment (MSA), Liu et al. proposed a tool MSA-CUDA (Liu et al., 2009b) to parallelize all three stages of ClustalW v2.0.9 processing pipeline by using inter-task parallelization. Although they also implemented the distance matrix calculation stage in MSA-CUDA by using intra-task parallelization, it only achieved about 10× speedups for 1000 sequences with length of 858 by comparing to ClustalW v2.0.9 on single-GPU. Lin and Lin proposed GPU-REMuSiC v1.0 (Lin and Lin, 2014) for constrained multiple sequence alignment by using intra-task parallelization to do the distance matrix calculation step on single- and multi-GPUs. They summarized the previous research for parallel dynamic programming algorithms on CPUs and then defined eight implementation types of dynamic programming on GPUs. The distance matrix calculation step in GPU-REMuSiC v1.0 adopted the Needleman and Wunsch algorithm to calculate the optimal similarity score, and it was implemented by the *Synchronous Row Multiple Threads* (*SRMT*) type.

Hence, in this work, we proposed a GPU version of ClustalW v2.0.11, called CUDA ClustalW v1.0, by using intra-task parallelization on single- and multi-GPUs. In CUDA ClustalW v1.0, the distance matrix calculation step was implemented by the *Synchronous Diagonal Multiple Threads* (*SDMT*) type (Lin and Lin, 2014). Moreover, several optimization methods were designed to improve the performance of CUDA ClustalW v1.0. From the experimental results, the CUDA ClustalW v1.0 can achieve about 22× speedups for 1000 sequences with length of 1523 in the distance matrix calculation step by comparing to ClustalW v2.0.11 on single-GPU. For the overall execution time, the CUDA ClustalW v1.0 can achieve about 33× speedups by comparing to ClustalW v2.0.11 on two-GPUs.

2. CUDA ClustalW v1.0

In general, the procedure of progressive multiple sequence alignment, such as ClustalW, can be divided into three steps: (1) the distance matrix calculation, (2) the guide tree creation, and (3) the progressive alignment. In distance matrix calculation step, the similarity scores were calculated from each pair of sequences by using the pairwise alignment algorithm, such as Needleman and Wunsch algorithm. For aligning two sequences both with length of n, the time and space complexity both are $O(n^2)$ by using Needleman and Wunsch algorithm. For k sequences in the progressive multiple sequence alignment, it needs to do $k^2/2$ pairwise alignments and the total time complexity is $O(k^2n^2)$. In guide tree creation step, a rooted or unrooted phylogenetic tree was built according to the calculated distance matrix by using phylogenetic tree construction algorithm. The time complexity of this step is $O(k^3)$ for most of phylogenetic tree construction algorithms. The built guide tree could be used to decide the orders of following progressive alignment step. In progressive alignment step, the required pairwise alignment results were generated according to the orders in guide tree. Although this step still needs to use pairwise alignment algorithm to obtain the pairwise alignment results, and then combines the previously generated alignments (a group of aligned sequences), the number of times of pairwise alignments in this step generally is far less than that in the distance matrix calculation step. Therefore, in most of cases, the computation time of distance matrix calculation step occupied more than 90 percent of overall execution time by progressive multiple sequence alignment. Hence, in CUDA ClustalW v1.0, we focused on the GPU implementation of distance matrix calculation step. In addition, several optimization methods by considering the memory usage, load balancing, and threads/thread blocks adjustment were designed to enhance the performance of CUDA ClustalW v1.0.

2.1. Distance matrix calculation step

As mentioned above, in distance matrix calculation step, the similarity scores were calculated from each pair of sequences by using pairwise alignment algorithm. The space complexity is $O(n^2)$ by using Needleman and Wunsch algorithm for aligning two sequences both with length of n. For real biological applications, the length of sequences may be large and the memory requirement cannot be met by a PC (CPU) or a graphic card (GPU). Therefore, in practice, the pairwise alignment algorithm used in ClustalW is consisted of three paths: forward path, reverse path, divide and conquer path. This idea is proposed by Hirschberg algorithm (Hirschberg, 1977) for solving the longest common subsequence problem. When using the dynamic programming algorithm to align two sequences, the process can be seen as the score calculations in a two-dimensional array. By using the Hirschberg algorithm, this two-dimensional array will be divided into forward array and reverse array equally. In the forward array, the forward path is to calculate the scores from left-top to right-down. Similarly, in the reverse array, the reverse path is to calculate the scores from right-down to left-top. The computations of the forward array and reverse array can be done concurrently. The dependency of dynamic programming algorithm in the forward path and the reverse path is shown in Fig. 1, respectively. In order to reduce the space complexity from $O(n^2)$ to $O(n)$, the entire forward array or reverse array could not be stored in memory. Only the scores in the last row of forward array and reverse array can be stored in memory, respectively. Hence, after doing the computations of the forward path and reverse path, the scores in the last row of forward array are merged (addition operation) with those of reverse array. Then a break point with the maximal score in an

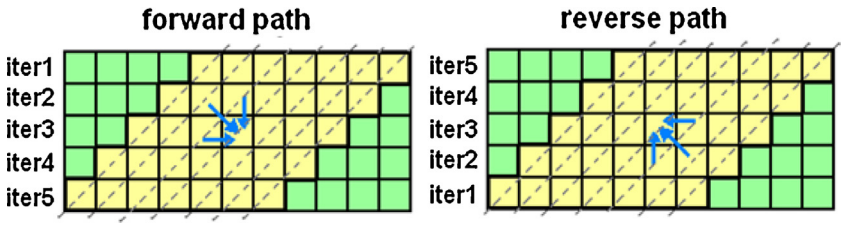

Fig. 1. The dependency of dynamic programming algorithm in the forward path and the reverse path, respectively.

optimal path could be found. This break point then is used to divide the original two-dimensional array into two sub-arrays according to the substitution, insertion or deletion cases (a pair of characters (include gap) aligned by two sequences). Afterward, each sub-array will be divided into forward array and reverse array again and repeats the above actions. Fig. 2 shows the concept of forward path and reverse path. While all break points in an optimal path are found, a divide and conquer path is used to calculate the similarity score of two sequences. The details of divide and conquer path can be found in the literatures (Korf and Zhang, 2000; Rajko and Aluru, 2004). In CUDA ClustalW v1.0, we follow the design of ClustalW. In order to do the computations in forward array and reverse array concurrently, the threads in a thread block are divided into two parts, one for forward array and another for reverse array. Only one thread in a thread block is then used to merge the scores in the last row of forward array and reverse array, find the break point, and divide the original two-dimensional array into two sub-arrays. In ClustalW, the implementation of divide and conquer path is done by the recursive function. Although newer CUDA version supports the recursive function, the implementation in CUDA ClustalW v1.0 is done by the loop function with the stacks in order to be used for the older CUDA version.

For k sequences in CUDA ClustalW v1.0, there are $k^2/2$ pairwise alignments in distance matrix calculation step. CUDA ClustalW v1.0 used intra-task parallelization to assign a pairwise alignment to a thread block. For a pairwise alignment in a thread block, there are two directions, row (column) and diagonal, of calculating scores in a two-dimensional array. According to the dependency of dynamic programming algorithm (Fig. 1), the computation of each array element in a diagonal is independent. When the maximal length of a diagonal is larger than the number of threads in a thread block, the computation of a diagonal can be asynchronous or synchronous. Since the implementation of asynchronous

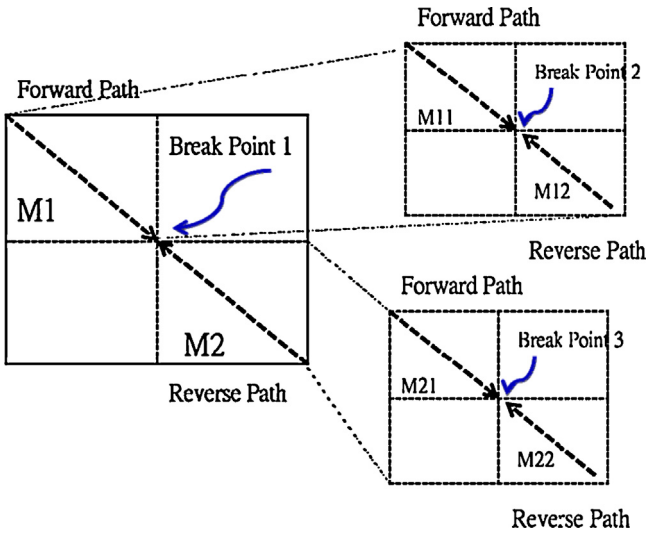

Fig. 2. The concept of forward path and reverse path.

computation in a thread block is difficult, the implementation of CUDA ClustalW v1.0 adopted the synchronous computation. Therefore, the distance matrix calculation step in CUDA ClustalW v1.0 was implemented by the *Synchronous Diagonal Multiple Threads* (*SDMT*) type (Lin and Lin, 2014). An example of using *SDMT* type to do the distance matrix calculation step is shown in Fig. 3. For a pairwise alignment, in the forward path or reverse path, the inputs are two sequences and a substitution matrix. The substitution matrix is used to determine the score of each pair of characters aligned by two sequences. The output is a similarity score. A simple way is to allocate the inputs and output into the global memory of GPU device. All of similarity scores from pairwise alignments were stored in a distance matrix located at the global memory.

2.2. Optimization methods for distance matrix calculation step

In CUDA ClustalW v1.0, the GPU implementation of distance matrix calculation step is based on the intra-task parallelization and *SDMT* type. For GPU, there are several memory architectures, hundreds cores, and large memory bandwidth. It is worth to improve the performance of CUDA ClustalW v1.0 by using these advantages. Several optimization methods were designed for distance matrix calculation step and evaluated in this work. All of these methods are summarized as follows.

2.2.1. Load balancing strategy

Using intra-task parallelization, a pairwise alignment is assigned to a thread block. Although the number of thread blocks can be declared large, the number of streaming multiprocessors in a graphic card is limited. In general, a streaming multiprocessor can deal with one thread block or more. The occupancy is determined by the shared memory usage and register usage. For k sequences in CUDA ClustalW v1.0, there are $k^2/2$ pairwise alignments. For a kernel function, it only deals with a fixed number M of pairwise alignments with the similar length of sequences. This number M is the several times of number of streaming multiprocessors. After finishing a kernel function, the similarity scores calculated are copied from device (GPU) memory into host (GPU) memory and then next kernel function is started.

2.2.2. Streaming technique

When using the load balancing strategy mentioned above, the input data for M pairwise alignments are copied from host memory into device memory at first, then the similarity scores are calculated by device, and finally the similarity scores are copied from device memory into host memory. By this way, the computation step should be delayed waiting for data transfer step. Therefore, the streaming technique, a pipeline of asynchronous data transmission, can be used to overlap the data transfer time and computation time.

2.2.3. Memory allocation

For a pairwise alignment, the inputs are two sequences and a substitution matrix. These two sequences should be accessed by

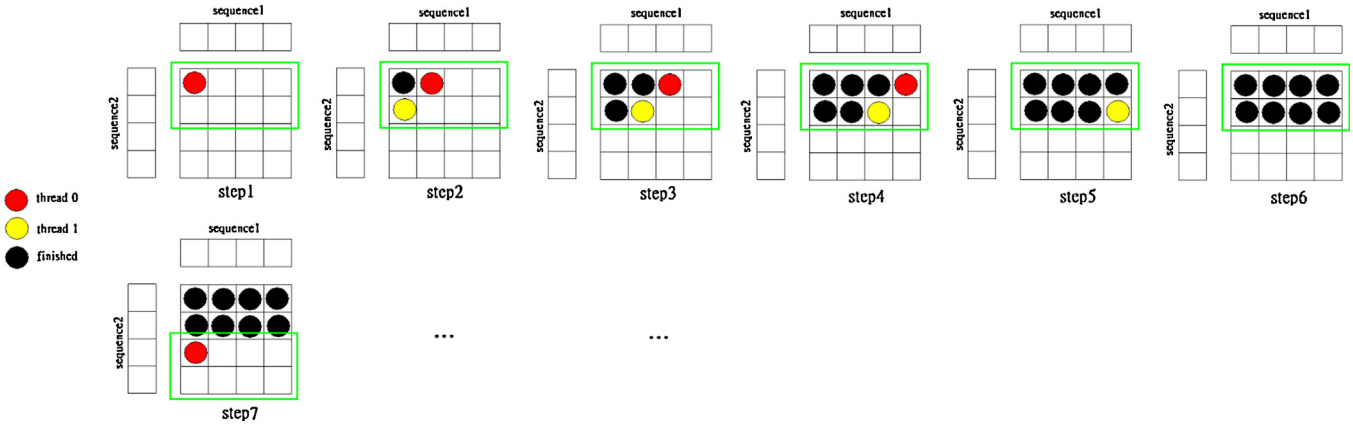

Fig. 3. An example of using SDMT type to do the distance matrix calculation step in CUDA ClustalW v1.0.

each thread in a thread block to query the substitution matrix. When input sequences are copied from host memory into device memory, each pair of sequences for a thread block are accessed and stored in the shared memory. Since the substitution matrix is read-only and the size of it is less than 4 k byte, it will be stored in the texture memory. The distance matrix is stored in the global memory.

2.2.4. Unsigned char data type

When each pair of sequences for a thread block are accessed and stored in the shared memory, the shared memory usage will affect the occupancy of streaming multiprocessors. Hence, the sequences are stored by using the *unsigned char* data type, not *int*, in order to reduce the shared memory usage. Although it may cause bank conflict, the performance can be improved obviously by increasing the occupancy of streaming multiprocessors.

2.2.5. Precision adjustment

When calculating the similarity scores on GPU, the used division operator is a single-precision floating-point operation. Since the single-precision floating-point operations in CUDA are not designed according to the IEEE-754 standard. The similarity scores calculated by CPU and GPU may be different. This result may cause the alignment results generated by CPU and GPU are inconsistent. In CUDA ClustalW v1.0, *_fdiv_rn*() function was used to replace the built-in division operator. It is worth to note that the goal of this optimization method is to avoid the precision problem by using division operator on GPU. We cannot guarantee that the alignment results by CUDA ClustalW v1.0 are the same with those by ClustalW for any test set. As mentioned in the literature (Vouzis and Sahinidis, 2011), the most of GPU implementations of BLAST are not guaranteed to give results identical to NCBI-BLAST. In the past, most of GPU implementations are done by rewriting the

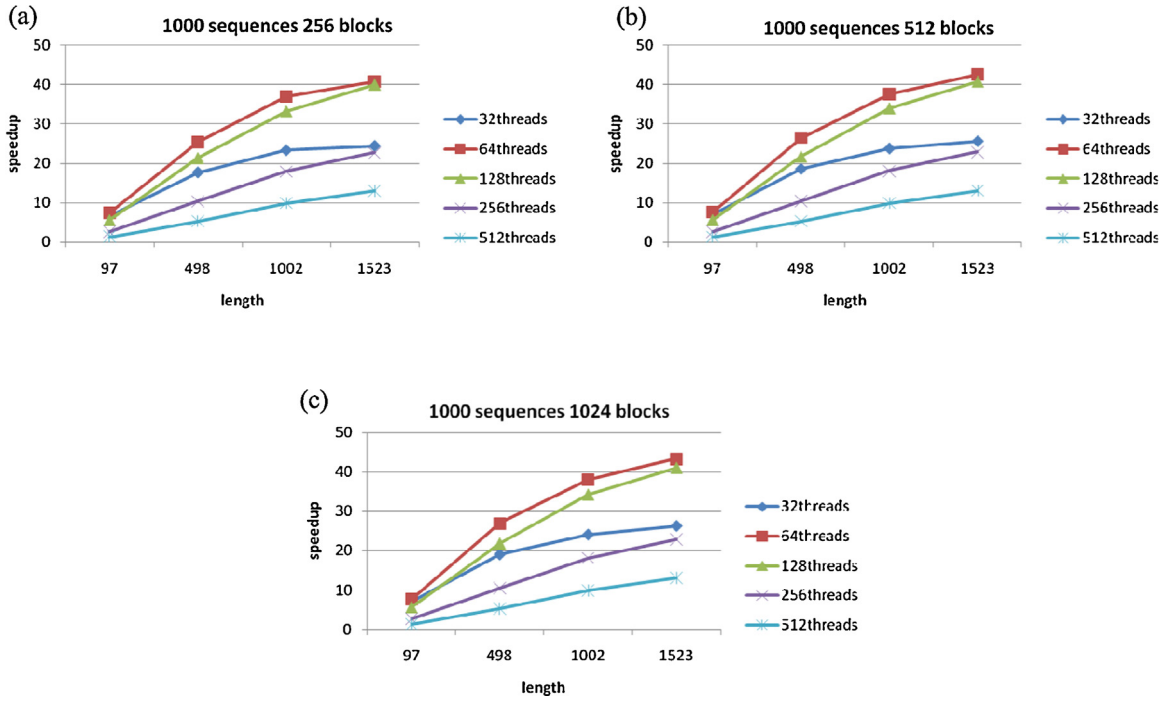

Fig. 4. Speedups of distance matrix calculation step by comparing CUDA ClustalW v1.0 with ClustalW v2.0.11 under various numbers of threads in a thread block and various numbers of thread blocks in a grid.

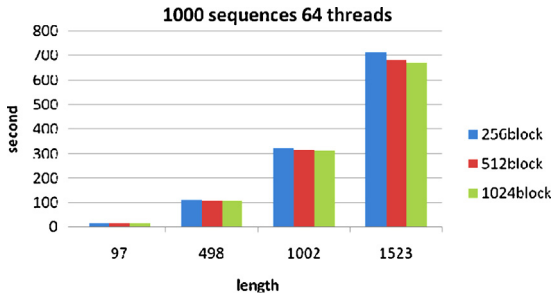

Fig. 5. Execution time of distance matrix calculation step by CUDA ClustalW v1.0 under the 64 threads in a thread block and various numbers of thread blocks in a grid.

programs according to the original algorithm. The result obtained by this way is very different to that by the original program. Few GPU implementations are done by modifying the necessary functions in the original program. Even by this way, the result by GPU implementation has slight difference by comparing with that by the original program (Vouzis and Sahinidis, 2011). In CUDA ClustalW v1.0, we shares many data structures with ClustalW and only modify the necessary functions in ClustalW.

2.2.6. Thread and thread block adjustment

On GPU, the number of threads in a thread block and the number of thread blocks in a grid will affect the performance of CUDA ClustalW v1.0. Hence, the CUDA ClustalW v1.0 was evaluated by various numbers of threads in a thread block and the number of thread blocks in a grid for test sequences in the experimental tests (see Section 3). The best choice is to set 64 and 1024 for the number of threads in a thread block and the number of thread blocks in a grid, respectively.

2.2.7. Multi-GPUs implementation

The CUDA ClustalW v1.0 can be used on Multi-GPUs by using OpenMP library. The number of graphic cards inserted in PCI-E will be detected by CUDA ClustalW v1.0 and then fully utilize their computing power. CUDA ClustalW v1.0 assumes a homogeneous environment and assigns fixed number of pairwise alignments to each graphic card.

3. Experimental tests

The CUDA ClustalW v1.0 was implemented and evaluated on a test platform. There are three versions for the CUDA ClustalW v1.0. The version 1 was implemented with optimization methods: *load balancing strategy*,*streaming technique*, *memory allocation*, *precision adjustment*, and *thread and thread block adjustment*. The version 2 was implemented under the version 1 with the optimization method: *unsigned char data type*. The version 3 (Final version) was implemented under the version 2 with the optimization method: *multi-GPUs implementation*. In the test platform, the host (CPU) is Intel Xeon X5550 2.67 GHz with 24GB DDRIII-1333 RAM running the CentOS v5.3 operating system. The devices (GPU) have two NVIDIA Tesla C2050 cards with kernel v2.6.18. The C compiler is gcc v4.1.2 and the CUDA compiler is nvcc v3.0. The ClustalW v2.0.11 was compiled with the option 'O2' and the CUDA ClustalW v1.0 was compiled with the options '-arch=sm_13' and '-use_fast_math'.

The test protein sequences were downloaded from the NCBI website (www.ncbi.nlm.nih.gov/), and these sequences can be classified into eight test sets: (1) 100 sequences with length of 97, (2) 100 sequences with length of 498, (3) 100 sequences with length of 1002, (4) 100 sequences with length of 1523, (5) 1000 sequences with length of 97, (6) 1000 sequences with length of 498, (7) 1000 sequences with length of 1002, (8) 1000 sequences with length of 1523. At first, the CUDA ClustalW v1.0 (Final version) was evaluated by various numbers of threads in a thread block and the number of thread blocks in a grid for test set-8. Fig. 4 shows the speedups of distance matrix calculation step by comparing CUDA ClustalW v1.0 with ClustalW v2.0.11 under various numbers of threads in a thread block and various numbers of thread blocks in a grid. In Fig. 4, the best speedups achieved by CUDA ClustalW v1.0 all are under the 64 threads in a thread block. Fig. 5 shows the execution time of distance matrix calculation step by CUDA ClustalW v1.0 for test set-8 under the 64 threads in a thread block and various numbers of thread blocks in a grid. From Fig. 5, the lowest execution time of distance matrix calculation step by CUDA ClustalW v1.0 all are under the 1024 thread blocks in a grid. The best choice is to set 64 and 1024 for the number of threads in a thread block and the number of thread blocks in a grid, respectively. The following tests all are based on this choice.

Tables 1 and 2 show the overall execution time by ClustalW v2.0.11 and CUDA ClustalW v1.0 on 100 and 1000 sequences, respectively. From Table 1, the execution time by CUDA ClustalW v1.0 is larger than that of ClustalW v2.0.11 for the test set-1. The reason is that the size of test data is too small. For other test sets, execution time by CUDA ClustalW v1.0 is all shorter than that of ClustalW v2.0.11. Besides, the performance by CUDA ClustalW v1.0 (version 2) can be improved greatly by comparing to its version 1. It shows that the occupancy of streaming multiprocessors can affect the performance. However, the performance of its final version is not improved obviously by comparing to its version 2 due to the small size of test data. For all of test sets in Table 2, the execution time by CUDA ClustalW v1.0 is all shorter than that of ClustalW v2.0.11. Moreover, the performance by CUDA ClustalW v1.0 (version 2) can be improved greatly by comparing to its version 1. Similarly, the performance by its final version can be improved greatly by comparing to its version 2. It shows that CUDA ClustalW v1.0 is useful for multi-GPUs. In the experimental tests, a program, bali_score, downloaded from the Balibase benchmark (Thompson et al., 1999) (http://www.lbgi.fr/balibase/) was used to compare the alignment results by CUDA ClustalW v1.0 with those by ClustalW v2.0.11. In bali_score program, the reference alignment is set to the alignment results by ClustalW v2.0.11, and then the alignment results by CUDA ClustalW v1.0 is the test alignment. Two scores, Sum-of-Pairs score (SP) and the Total-Column score (TC),

Table 1
Overall execution time by ClustalW v2.0.11 and CUDA ClustalW v1.0 on 100 sequences.

	Execution time (second)			
	Sequence Length			
Tool	97	498	1002	1523
ClustalW v2.0.11	1.370	32.090	142.690	328.800
CUDA ClustalW v1.0 (version 1)	3.489	9.182	23.083	45.846
CUDA ClustalW v1.0 (version 2)	3.159	7.193	18.303	36.936
CUDA ClustalW v1.0 (Final version)	4.317	8.031	18.256	35.569

Table 2
Overall execution time by ClustalW v2.0.11 and CUDA ClustalW v1.0 on 1000 sequences.

	Execution time (second)			
	Sequence Length			
Tool	97	498	1002	1523
ClustalW v2.0.11	131.200	2882.440	12020.400	29320.200
CUDA ClustalW v1.0 (version 1)	67.846	435.723	1177.930	2456.230
CUDA ClustalW v1.0 (version 2)	38.397	237.146	711.489	1559.380
CUDA ClustalW v1.0 (Final version)	29.335	136.594	408.880	892.664

Table 3
SP and TC scores by ClustalW v2.0.11 and CUDA ClustalW v1.0 on 100 and 1000 sequences.

Number of sequences	Sequence length	SP score	TC score
100	97	0.99	1
	498	1	1
	1002	1	1
	1523	1	1
1000	97	1	1
	498	1	1
	1002	1	1
	1523	1	1

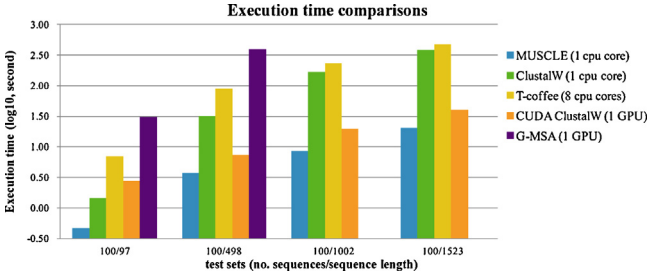

Fig. 7. Overall execution time (represented by log10) by these tools on four test sets.

were used to estimate the accuracy of a test alignment comparing to reference alignment. The maximal values of these two scores both are 1, means that the test alignment is identically to the reference alignment in aligning. Table 3 shows the SP and TC scores by CUDA ClustalW v1.0 on these eight test sets. From Table 3, we can observe that most of alignment results of CUDA ClustalW v1.0 are identical to those of ClustalW v2.0.11, even though the alignment results of minor cases are not identical, they are also highly similar.

Fig. 6 shows the speedups of distance matrix calculation step by comparing CUDA ClustalW v1.0 with ClustalW v2.0.11. From Fig. 6, the speedup increases when the size of test data increases. On single-GPU, CUDA ClustalW v1.0 can achieve 22× speedups by comparing to ClustalW v2.0.11. The speedup by CUDA ClustalW v1.0 can be double (43×) on two-GPUs. For overall execution time, CUDA ClustalW v1.0 can achieve 19× and 33× speedups by comparing to ClustalW v2.0.11 on single- and two-GPUs. T-coffee (Notredame et al., 2000b), MUSCLE (Robert, 2004), and MAFFT (Katoh et al., 2002) are also famous multiple sequence alignment tools. In general, none of these tools and ClustalW dominates the field of multiple sequence alignment. By considering the computation time of these tools, MUSCLE is the fastest tool and T-coffee is the slowest tool among them. G-MSA (Blazewicz et al., 2013) is a GPU version of T-coffee on multi-GPUs. Hence, the CUDA ClustalW v1.0 also was used to compare with them. T-coffee v10.00.r1613 and MUSCLE v3.8.425 both were downloaded from the EMBL-EBI website (http://www.ebi.ac.uk/). G-MSA was downloaded from the homepage listed in the literature. The downloaded T-coffee will detect the number of CPU cores and then fully utilize their computing power. Another test platform was used to test CUDA ClustalW v1.0, G-MSA, ClustalW v2.0.11, T-coffee v10.00.r1613, and MUSCLE v3.8.425. In this test platform, the host is Intel i7CPU 920 2.67 GHz (8 cores) with 6GB RAM running the Linux version 3.0.0-12-generic operating system. The device is one NVIDIA Tesla K20c card. The C compiler is gcc v4.4.6 and the CUDA compiler is nvcc v5.0. The test sets 1-4 above were also used to evaluate these four tools. Fig. 7 shows the overall execution time (represented by $\log_{10}$) by using these tools with four test sets. Due to the constraint in G-MSA, the test sets (100/1002) and (100/1523) cannot be aligned by G-MSA. From Fig. 7, MUSCLE v3.8.425 also is the fastest tool and T-coffee v10.00.r1613 is the slowest tool. However, for large size of test sets, the execution time by CUDA ClustalW v1.0 is close to that of MUSCLE v3.8.425. This result expands the range of applications by using ClustalW.

4. Conclusion

In this study, CUDA ClustalW v1.0 was proposed. CUDA ClustalW v1.0 used intra-task parallelization and *SDMT* type to implement the distance matrix calculation step. Moreover, several optimization methods were designed and implemented in CUDA ClustalW v1.0 by considering the memory usage, load balancing,

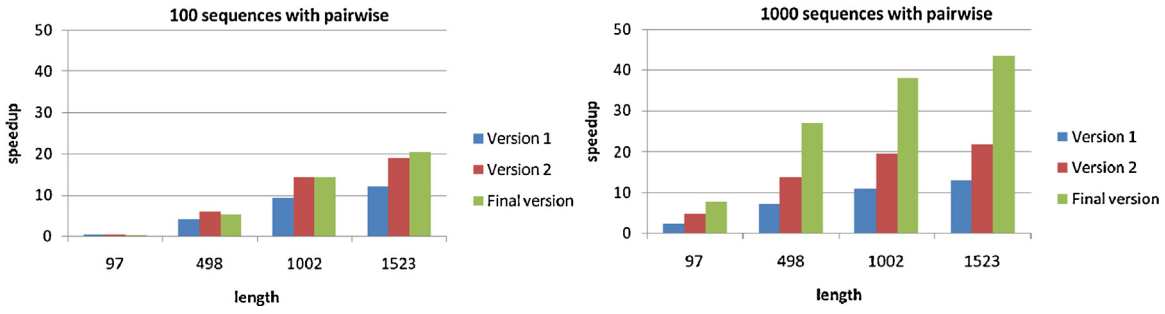

Fig. 6. Speedups of distance matrix calculation step by comparing CUDA ClustalW v1.0 with ClustalW v2.0.11.

and threads/thread blocks adjustment. The experimental results showed that the CUDA ClustalW v1.0 could achieve satisfied speedups by comparing with ClustalW v2.0.11 for a large number of long sequences. The CUDA ClustalW v1.0 could be included into other multiple sequence alignment tools. The software of CUDA ClustalW v1.0 and the test sets can be downloaded at http://140.114.91.64/cuda-clustalw/.

Acknowledgments

The authors would like to thank the National Science Council of the Republic of China, Taiwan, for partially supporting this research under Contract Nos. NSC100-2221-E-126-007-MY3, NSC100-2221-E-182-057-MY3, and NSC103-2632-E-126 -001-MY3.

References

Blazewicz, J., Frohmberg, W., Kierzynka, M., 2013. P.l Wojciechowski G-MSA-A GPU-based, fast and accurate algorithm for multiple sequence alignment. J. Parallel Distrib. Comput. 73, 32–41.
Carrillo, H., Lipman, D.J., 1988. The multiple sequence alignment problem in biology. SIAM J. Appl. Math. 48, 1073–1082.
Feng, D.F., Doolittle, A.F., 1987. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. J. Mol. Evol. 25, 351–360.
Hirschberg, D.S., 1977. Algorithm for the longest common subsequence problem. J. ACM 24 (4), 664–675.
Katoh, K., Misawa, K., Kuma, K., Miyata, T., 2002. MAFFT: a novel method for rapid multiple sequence alignment based on fast Fourier transform. Nucleic Acids Res. 30, 3059–3066.
Khajeh-Saeed, A., Poole, S., Perot, J.B., 2010. Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors. J. Comput. Phys. 229 (11), 4247–4258.
Korf, R.E., Zhang, W., 2000. Divide and conquer frontier search applied to optimal sequence alignment. Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence 910–916.
Lee, S.T., Lin, C.Y., Hung, C.L., 2013. GPU-Based cloud service for Smith–Waterman algorithm using frequency distance filtration scheme. BioMed Res. Int. 2013 Article ID 721728.
Li, K.B., 2003. ClustalW-MPI: ClustalW analysis using distributed and parallel computing. Bioinformatics 19 (12), 1585–1586.
Lin, C.Y., Lin, Y.S., 2014. Efficient parallel algorithm for multiple sequence alignments with regular expression constraints on graphics processing units. Int. J. Comput. Sci. Eng. 9 (1/2), 11–20.
Liu, Y., Maskell, D.L., Schmidt, B., 2009a. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. BMC Res. Notes 2, 73.
Liu, Y., Schmidt, B., Maskell, D.L., 2009b. MSA-CUDA: multiple sequence alignment on graphics processing units with CUDA. ASAP 121–128.
Liu, Y., Schmidt, B., Liu, W., Maskell, D.L., 2010a. CUDA–MEME: accelerating motif discovery in biological sequences using CUDA-enabled graphics processing units. Pattern Recogn. Lett. 31, 2170–2177.
Liu, Y., Schmidt, B., Maskell, D.L., 2010b. CUDASW++2.0: enhanced Smith–Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. BMC Res. Notes 3, 93.
Liu, W., Schmidt, B., Müller-Wittig, W., 2011. CUDA-BLASTP: accelerating BLASTP on CUDA-enabled graphics hardware. IEEE/ACM Trans. Comput. Biol. Bioinf. 8 (6), 1678–1684.
Manavski, S.A., Valle, G., 2008. CUDA compatible GPU cards as efficient hardware accelerators for Smith–Waterman sequence alignment. BMC Bioinf. 9 (Suppl 2), S10.
Needleman, S.B., Wunsch, C.D., 1970. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J. Mol. Biol. 48, 443–453.
Nickolls, J., Buck, I., Garland, M., Skadron, K., 2008. Scalable parallel programming with CUDA. ACM Queue 6, 40–53.
Notredame, C., Higgins, D.G., Heringa, J., 2000a. T-Coffee: a novel method for fast and accurate multiple sequence alignment. J. Mol. Biol. 302, 205–217.
Notredame, C., Higgins, D.G., Heringa, J., 2000b. T-Coffee: a novel method for fast and accurate multiple sequence alignment. J. Mol. Biol. 302, 205–217.
Rajko, S., Aluru, S., 2004. Space and time optimal parallel sequence alignments. IEEE Trans. Parallel Distrib. Syst. 15 (12), 1070–1081.
Robert, C.E., 2004. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Res. 32 (5), 1792–1797.
Sandes, F.D.O., Melo, A.C.M.A.D., 2010. CUDAlign: using GPU to accelerate the comparison of megabase genomic sequences. PPOPP 137–146.
Sandes, F.D.O., Melo, A.C.M.A.D., 2011. Smith–Waterman alignment of huge sequences with gpu in linear space. IPDPS 1199–1211.
Schatz, M.C., Trapnell, C., Delcher, A.L., Varshney, A., 2007. High-throughput sequence alignment using Graphics Processing Units. BMC Bioinf. 8, 474.
Smith, T.F., Waterman, M.S., 1981. Identification of common molecular subsequences. J. Mol. Biol. 147, 195–197.
Thompson, J.D., Higgins, D.G., Gibson, T.J., 1994. CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. Nucleic Acids Res. 22 (22), 4673–4680.
Thompson, J.D., Plewniak, F., Poch, O., 1999. BAliBASE: a benchmark alignment database for the evaluation of multiple alignment programs. Bioinformatics 15, 87–88.
Trapnell, C., Schatz, M.C., 2009. Optimizing data intensive GPGPU computations for DNA sequence alignment. Parallel Comput. 35, 429–440.
Vouzis, P.D., Sahinidis, N.V., 2011. GPU-BLAST: using graphics processors to accelerate protein sequence alignment. Bioinformatics 27 (2), 182–188.
Wang, L., Jiang, T., 1994. On the complexity of multiple sequence alignment. J. Comput. Biol. 1, 337–348.
Wong, K.C., Zhang, Z., 2014. SNPdryad: predicting deleterious non-synonymous human SNPs using only orthologous protein sequences. Bioinformatics 30 (8), 1112–1119.
Wong, K.C., Chan, T.M., Peng, C., Li, Y., Zhang, Z., 2013. DNA motif elucidation using belief propagation. Nucleic Acids Res. 41 (16), e153.