Implementation of CUDA GPU-Based Parallel Computing on Smith-Waterman Algorithm to Sequence Database Searches

Alhadi Bustamam, Gianinna Ardaneswari, Dian Lestari
Department of Mathematics, Universitas Indonesia
Email: alhadi@sci.ui.ac.id, ninna@sci.ui.ac.id, dian.lestari@sci.ui.ac.id

***Abstract*— In bioinformatics, one of the gold-standard algorithms to compute the optimal similarity score between sequences in a sequence database searches is Smith-Waterman algorithm that uses dynamic programming. This algorithm has a quadratic time complexity which requires a long computation time for large-sized data. In this issue, parallel computing is essential for sequence database searches in order to reduce the running time and to increase the performance. In this paper, we discuss the parallel implementation of Smith-Waterman algorithm in GPU using CUDA C programming language with NVCC compiler on Linux environment. Furthermore, we run the performance analysis using three parallelization models, including Inter-task Parallelization, Intra-task Parallelization, and a combination of both models. Based on the simulation results, a combination of both models has better performance than the others. In addition the parallelization using combination of both models achieves an average speed-up of 313× and an average efficiency with a factor of 0.93.**

I. INTRODUCTION

IN bioinformatics, sequence database searches can be applied to find the similarity between a query sequence (the sequence to be analyzed) and subject sequences (the well known sequence) in a sequence database. Such information can be used for further research, for example to identify evolutionary relationships in a species [1-4]. The similarity of two sequences can be identified through the similarity scores obtained from sequence alignment. Sequence alignment is the process of arranging sequence to identify the similarity of two or more sequences. The alignment of two sequences is referred to as a pair-wise alignment. One of the popular algorithms which is capable to produce the optimal similarity score is Smith-Waterman algorithm that uses dynamic programming [4-7]. Due to exponential growth of sequence databases, the algorithm and computational tools or media must be devised to improve the performance of sequence database searches [8][9].

Smith-Waterman algorithm was first introduced by T. F. Smith and M. S. Waterman in 1981 and further enhanced by O. Gotoh in 1982. This algorithm is one of the most widely used algorithms to compute and locate sequence similarity in the biological sequence databases. Unfortunately, this algorithm requires a large computational complexity $O(n^2)$ [8]. In addition, the rate of growth in terms of their number and size of data of DNA or protein sequence databases is exponential, resulting Smith-Waterman algorithm performance is so slow in a very large sequence data which make it, to a certain extent, unrealistic to apply in those circumstances. Therefore, parallel programming techniques is essential as a solution to these problems.

Classic parallel programming based on distributed or cluster computing with MPI (Message Passing Interface) as a common parallel programming interface often encounters several obstacles, such as not scalable. Moreover, its implementation processes are labor-intensive and parallelization tasks of the program are developed manually (hard), and for most cases it also requires a great amount of resources [9][10]. The current technology of parallel computing on the GPU (Graphics Processing Unit) develops very rapidly. GPU implement many-core architectures that consisting of thousands of threads in hundreds of cores, in contrast to the CPU (Central Processing Unit), which implement multi-core architectures that consisting multiples of 2 cores. In GPU computing,

Manuscript received July 10, 2013; revised September 13, 2013, accepted September 14, 2013. This work was supported in part by the "Hibah Riset Unggulan Perguruan TInggi (BOPTN) 2013"DGHE INDONESIA, under Grant No.1568.

A. Bustamam is with the Department of Mathematics, Universitas Indonesia, Depok, 16424 INDONESIA (e-mail: alhadi@ sci.ui.ac.id).

G. Ardaneswari is with the Department of Mathematics, Universitas Indonesia, Depok, 16424 INDONESIA (e-mail: ninna@ sci.ui.ac.id).

D. Lestari is with the Department of Mathematics, Universitas Indonesia, Depok, 16424 INDONESIA (e-mail: dian.lestari@ sci.ui.ac.id).

with thousands of threads in hundreds of cores on the GPU, massively parallel programming can be applied to obtain substantial performance improvements [9][11][12].

In 2007, NVIDIA issued a standard model of parallel programming on GPUs called the CUDA (Compute Unified Device Architecture). CUDA is a minimal extension of the programming languages C / C + + to drive the process from the CPU to GPU computing. Since the CUDA, the GPU CUDA-based parallel programming into one effective solution over the obstacles encountered in the implementation of MPI [9][13].

Since the GPU contains a grid which consists of blocks with many threads inside them, this parallel computing on the GPU can be done in three models, namely the inter-task parallelization, intra-task parallelization, and a combination of both models. Inter-task parallelization is parallelization models with each task assigned to exactly one thread to be run separately in parallel. Intra-task parallelization is parallelization models with each task assigned to exactly one thread block to be executed separately and in parallel [8][14].

This paper evaluates separately the performance of these three models of parallel implementations for Smith-Waterman algorithm on the NVIDIA GeForce GTX460 GPU and Intel® Dual Core™ E5500 2.8GHz CPU. The implementation of each model based on [8] and CUDASW++ from [15]. We use the latest and larger protein sequences compare to datasets on [8]. The protein sequences data were taken from the Swiss-Prot database via online website at www.ebi.ac.uk/swissprot/. Calculation of similarity scores between two sequences is using the Smith-Waterman algorithm with the BLOSUM62 scoring matrix [16].

II. Parallel Approaches for Smith-Waterman Algorithm

Smith-Waterman algorithm is a local alignment and one of the gold-standard algorithms for computing the optimal similarity score between two sequences using dynamic programming. Suppose that two sequences are $A = a_1a_2 \ldots a_n$ and $B = b_1b_2 \ldots b_m$. Matrix similarity score H (with initialization $H_{k0} = H_{0l} = 0$ for $0 \le k \le n$ and $0 \le l \le m$) is computed using Equation 1 as the following:

$$H_{i,j} = max \begin{cases} max\{H_{i-1,j-1} + s(a_i, b_j), 0\} \\ max_{0<k<i}\{H_{i-k,j} - (G_s + kG_e)\} \\ max_{0<l<j}\{H_{i,j-l} - (G_s + lG_e)\} \end{cases} \quad (1)$$

for $1 \le i \le n$ and $1 \le j \le m$ where G_s is gap start-penalty and G_e is gap extension-penalty.

If Eq. (1) is naively implemented, then the column maximum and row maximum (second and third items in Eq. (1)) are repetitively calculated. This is alleviated by retaining the previous row and column sums, called $F(i,j)$ and $E(i,j)$. This increases the storage required by the algorithm by the factor of 3 but reduces the work significantly [17]. Therefore, the formula in the Smith-Waterman algorithm is as follows.

Suppose given two sequences, namely S_a and S_b with length l_a and l_b, Smith-Waterman algorithm calculates each cell of the similarity score matrix $H(i, j)$ of two sequences at positions i and j of S_a and S_b. Calculating $H(i, j)$ is based on Equation (2), for $1 \le i \le l_a$ and $1 \le j \le l_b$.

$$E(i,j) = max\{E(i,j-1) - G_e, H(i,j-1) - G_s - G_e\}$$
$$F(i,j) = max\{F(i-1,j) - G_e, H(i-1,j) - G_s - G_e\}$$
$$H(i,j) = max\{0, E(i,j), F(i,j), H(i-1,j-1) + s(a_i, b_j)\} \quad (2)$$

The initialization values for $H(i,j)$ are given as $H(i,0) = H(0,j) = E(i,0) = F(0,j) = 0$ for each index of $0 \le i \le l_a$ and $0 \le j \le l_b$. The maximum value of the local alignment (called *maxScore*) is defined as the maximum value of the matrix H (called the degree of similarity between sequences S_a and S_b).

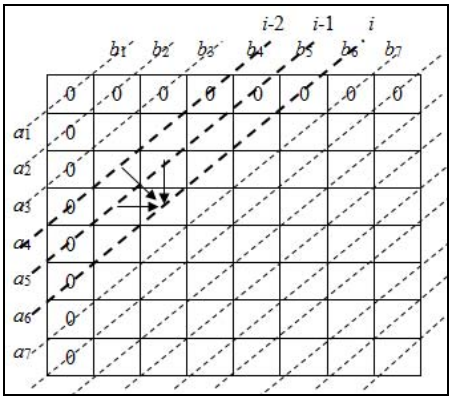

Fig 1. Data Dependency on the matrix H in the Smith-Waterman Algorithm [8]

Figure 1 shows the data dependency in the Smith-Waterman algorithm is based on Equation (2) for two sequences $A = a_1\, a_2\, a_3\, a_4\, a_5\, a_6\, a_7$ and $B = b_1\, b_2\, b_3\, b_4\, b_5\, b_6\, b_7$. In Figure 1, three arrows are showing the data dependencies in the matrix H, each cell depends on its left, upper, and upper-left neighbors. This dependency implies that all cells on the same minor diagonal in the matrix H are independent from each other and can be computed in parallel. Note that, in order to calculate minor diagonal i only the results of the minor diagonal (i-1) and (i-2). It became the basis of a strategy to parallelize the Smith-Waterman algorithm [8].

Parallelization of the Smith-Waterman algorithm using CUDA GPU-based parallel computing for the sequence database searches here is divided into three models, namely Inter-task parallelization, Intra-task parallelization, and a combination of both models. These three models will be discussed in more details in the following sub-sections.

A. Inter-task Parallelization

Inter-task parallelization is parallelization model with each task assigned to exactly one thread run separately and in parallel. In this study, Inter-task parallelization is defined as a model of parallelization where each thread will compute the similarity score between the query sequence and sequences of different subjects separately and in parallel (see Figure 2).

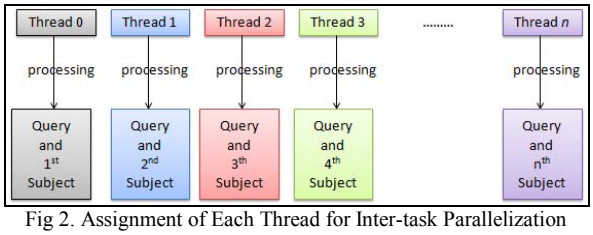

Fig 2. Assignment of Each Thread for Inter-task Parallelization

In order to maximize its performance and to reduce the bandwidth demand of global memory, this model use the cell block division method, where the alignment matrix is divided into cell blocks of equal size. A cell block is a square matrix of size *n* x *n*. Since one global memory access takes hundreds of clock cycles, the cell block division method leads to a signification reduction of the total runtime due to a reduction in the global memory accesses. However, the size of cell block is limited by the number of registers available per thread. Therefore, this leads to the optimal cell block size of 8 x 8 for this implementation [8].

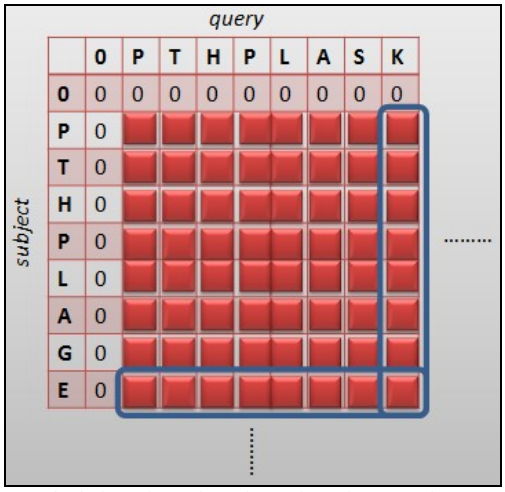

Fig 4. Calculation of Matrix *H* for 8 First Characters in Sequence

The input of a query sequence and a subject sequence are formed as the cell block size of 8 x 8 (as shown in Figure 4). If there are unused cells on the cell block, the cell are filled with a dummy value of zero so it does not change the score of similarity. At the first step, a matrix scoring is loaded into shared memory so that each thread can access the scoring matrix. In the second step, calculate index of each subject sequence in the existing threads and store the coordinates of the location and the length of each sequence. In the third step, look up the hash table to find the coordinates of the location and length of the subject sequence [8].

Similarity score calculations are performed on the fourth step. On each thread will be the Smith-Waterman algorithm calculation, which calculates the value of each cell in the matrix H and searches the degree of similarity. The calculation is performed for the first 8 characters in the query and subject sequences (see Figure 4). Value in the last row and column will be stored to use in the calculation of the next 8 characters. After calculating the whole values of the matrix *H*, the maximum value is used as the degree of similarity. Finally, at the last step, each thread will have a degree of similarity for each of the query sequence and subject sequences from the sequence database.

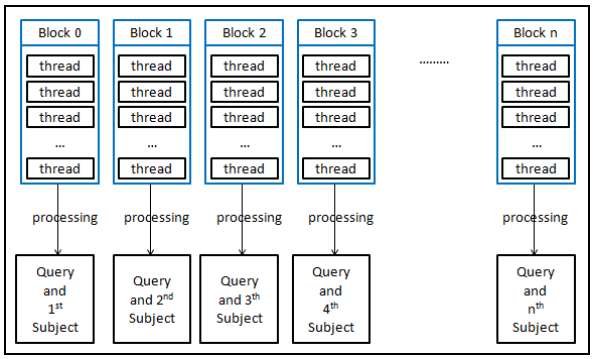

Fig 5. Assignment of Each Thread Block for Intra-task Parallelization

B. Intra-task Parallelization

Intra-task parallelization is parallelization model with each task assigned to exactly one thread block and executed separately in parallel. In this study, intra-task parallelization is defined as a model of parallelization where each thread block will compute the similarity score between the query sequence and sequences of different subjects separately in parallel (see Figure 5).

Based on Smith-Waterman algorithm (see Equation (2)), all cells on the same minor diagonal in the matrix *H* are independent from each other and can be computed in parallel. Note that, in order to calculate minor diagonal *i* only the results of the minor diagonal (*i*-1) and (*i*-2) (as shown in Figure 1). So, for Intra-task parallelization, thread block will calculate the similarity score matrix *H* for the query sequence and a subject sequence, then each thread in the block do the calculations for each cell on the diagonal minor *i* [8][15].

In the Intra-task parallelization model, calculation process of Smith-Waterman algorithm will be done in each thread block, which calculates the value of the matrix *H* and searches the degree of similarity (see Figure 7). The similarity value will be calculated on the *i*-th minor diagonal as the following: the red cell

will be computed by thread 0, blue cell will be computed by thread 1, green cell will be computed by thread 2, purple cell will be computed by the thread 3, and the last orange cell will be calculated by thread 4. The process is carried out in parallel. After calculation the *i*-th minor diagonal is finished, then the calculation is carried out for (*i*+1)th, (*i*+2)th minor diagonals, and so on. After calculating the whole values of the matrix *H*, the maximum value is used as the degree of similarity. At the last step, each thread block will have a degree of similarity for each of the query sequence and subject sequences from the sequence database.

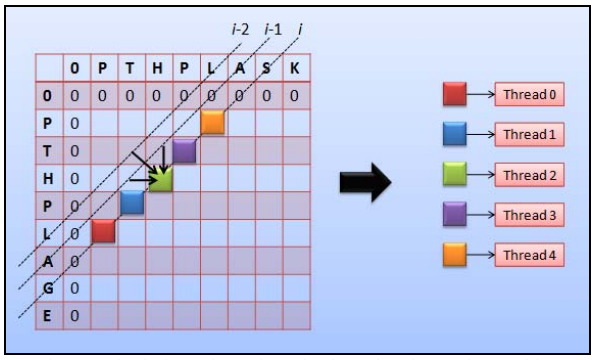

Fig 7. Calculation of matrix *H* in the *i*-th Minor Diagonal by Thread Block

C. Combination of Both Models Inter-task and Intra-task Parallelization

In the third model of parallelization, we use a combination of the two previous models, Inter-task parallelization and Intra-task parallelization. The computation processes are separated using the threshold value. If the subject sequence length is less than the threshold, then the calculation run by Inter-task parallelization, whereas if the subject sequence length greater than or equal to the threshold calculation run by Intra-task parallelization. The use of this threshold due to limitation of available local memory at each thread in the Inter-task parallelization model so it is more suitable to be applied to short sequences, otherwise Intra-task parallelization models which use shared memory with a capacity greater than local memory is more suitable to be applied for long sequences.

The selection of the threshold for the combination model of Inter-task parallelization and Intra-task parallelization depends on which sequence database we used. In this study, we use the Swiss-Prot 2013_4 sequence database. On this sequence database, we found that sequences with length less than 3072 have no significantly differences in their length. Meanwhile, the sequence with length greater than 3072 has a more wide difference in their length. Thus, 3072 is chosen as a threshold to combine the two models. After the selection of the threshold, it is found that there are 539,168 sequences in the sequence database which has a length of approximately 99.87% less than the threshold.

III. IMPLEMENTATION

In these implementations, we used some query sequences including protein sequences P04141, Q3K118, Q573C8, P12223, Q6ZWJ1, P75062, P09811, P27895, and P04775 with sequence length of 144, 189, 246, 464, 553, 657, 850, 1000, and 2005, respectively.

A. Running Time

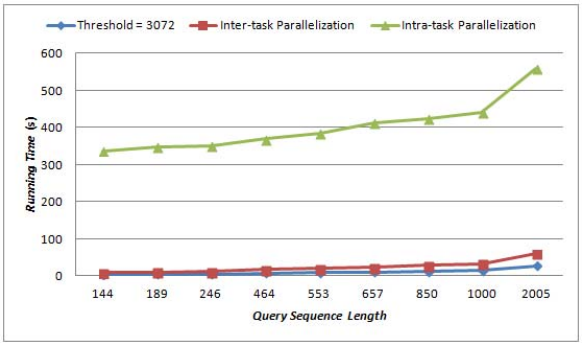

Fig 8. Runtime Comparison of CUDA GPU-Based Implementations for Smith-Waterman Algorithm

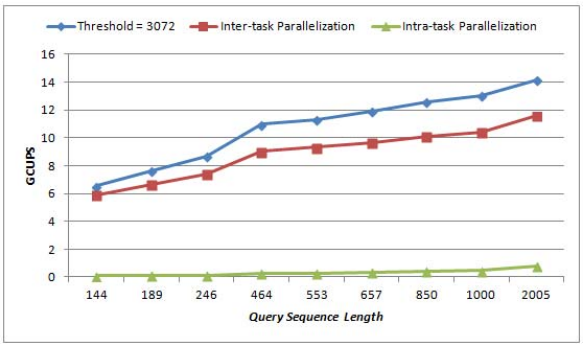

Fig 9. GCUPS Comparison CUDA GPU-Based Implementations of Smith-Waterman Algorithm

Based on the implementation results (see Figure 8), the parallel program that combines two models using threshold requires less runtime than the parallel program using solely Inter-task parallelization model or Intra-task parallelization model. Meanwhile, the program uses threshold has GCUPS larger than the other two models (see Figure 9). These results show the benefit of the combining parallelization program using the threshold since it takes the advantages of both of Inter-task and Intra-task parallelization models.

Table 1 shows the running time (in second and GCUPS) for all the query sequences running on CUDA Smith-Waterman implementations using Inter-task parallelization, Intra-task parallelization, and the combination of both models using threshold.

TABLE 1. RUNTIME AND GCUPS OF CUDA GPU-BASED IMPLEMENTATIONS OF SMITH-WATERMAN ALGORITHM

Query Sequence	Query Sequence Length	Threshold = 3072		Inter-task Parallelization		Intra-task Parallelization	
		Time (s)	GCUPS	Time (s)	GCUPS	Time(s)	GCUPS
P04141	144	4.212	6.553	4.676	5.903	328.114	0.084
Q3K118	189	4.740	7.643	5.443	6.656	337.724	0.107
Q573C8	246	5.421	8.699	6.368	7.404	339.470	0.139
P12223	464	8.095	10.986	9.851	9.028	350.948	0.253
Q6ZWJ1	553	9.364	11.319	11.369	9.323	363.692	0.291
P75062	657	10.573	11.910	13.055	9.646	387.999	0.325
P09811	850	12.958	12.573	16.129	10.101	394.478	0.413
P27895	1000	14.692	13.046	18.443	10.393	407.504	0.470
P04775	2005	27.172	14.143	33.176	11.584	497.248	0.773

TABLE 2. SPEED-UP AND EFFICIENCY OF CUDA GPU-BASED IMPLEMENTATION OF SMITH-WATERMAN ALGORITHM

Query Sequence	*Query Sequence Length*	*Runtime* (s)		*Speed-up*	Efficiency
		Sequential Program	Parallel Program		
Q6GZW1	119	1207,89	3,986	303,03	0,90
Q6GZW8	128	1261,47	4,079	309,26	0,92
Q6GZW5	137	1312,88	4,250	308,91	0,91
Q197D0	144	1389,45	4,259	326,24	0,97
Q6GZU1	153	1431,04	4,479	319,49	0,95
Average				313,39	0,93

In the Inter-task parallelization, each thread directly processes one subject sequence. Since each thread has limited available memory (local memory). Thus, this model is more suitable to be applied to short sequences. Meanwhile, in Intra-task parallelization, each thread block processes only one subject sequence, where each thread in the block did parallelization at each minor diagonal in matrix *H*. Moreover, the blocks have many available memory (shared memory), which allowing this model more suitable to be applied for long sequences. Therefore, by combining these two models Inter-task parallelization and Intra-task parallelization using a threshold yield better performance than using solely the Inter-task parallelization model or Intra-task parallelization model.

B. Speed-up and Efficiency

Speed-up is defined as the ratio between the running time of sequential program and the running time of parallel program. Efficiency is defined as a ratio between the speed-up and the number of processors used in parallelization process. In this paper we compute the speed-up and efficiency for parallel Smith-Waterman algorithm using a threshold by using the query sequence with length of 144 and the number of GPU processors of 336 processors.

To perform the speed-up and efficiency tests, we used some query sequences such as Q6GZW1, Q6GZW8, Q6GZW5, Q197D0, and Q6GZU1 with sequence length of 119, 128, 137, 144, and 153, respectively. Based on simulation results as shown in Table 2 we obtained an average speed-up at a factor of 313 times faster and the average efficiency at 0.93 (93% efficient). Overall, for the performance results shows a very significant improvement of CUDA GPU Implementation of Smith-Waterman algorithm using threshold model. Moreover, we can use more query sequences with varying lengths of sequences in order to obtain better and more accurate performance analysis results.

IV. CONCLUSION

In this paper, we discuss the application of parallel computing on the GPU CUDA-based Smith-Waterman algorithm for sequence database searches. Basically, the parallelization approaches can be done in three models, namely the Inter-task parallelization Intra-task parallelization, and combination of both Inter-task and Intra-task parallelization models.

Inter-task parallelization occupies less memory but achieve better performance than the Intra-task parallelization, for the short sequences. Meanwhile the Intra-task parallelization model requires more memory and more appropriate for long sequences. In order to get better performance we can use a combination of both models using threshold system. In this combination model, the subject sequences with their length less than the threshold will be compute using Inter-task parallelization, in contrast to the subject sequence of length greater than or equal to the threshold will be executed using Intra-task parallelization.

Based on our simulations, the implementation using a threshold system produces far better performance compare to the program which uses solely Inter-task parallelization model or Intra-task parallelization model. Moreover, the GPU-CUDA based parallel Smith-Waterman algorithm using a threshold, in our test obtained an average speed-up of 313× and an average efficiency with a factor of 0.93. This results show a very significant performance improvement for parallel Smith-Waterman algorithm implementation on GPU using CUDA.

ACKNOWLEDGMENT

This work is also support in part by DRPM UI, and PUSMAKA Laboratory at Department of Mathematics UI.

REFERENCES

[1] Isaev, A., *Introduction to Mathematical Methods in Bioinformatics*. Berlin: Springer-Verlag, 2004.

[2] Albert, B., Johnson, A., Lewis, J., Raff, M., Roberts, K., and Walter, P., *Molecular Biology of Cell*, 4th ed., New York: Garland Science, 2002.

[3] Bayat, A., "Bioinformatics: Science, Medicine, and The Future," *BMJ*, vol. 324, pp.1018-1022, April 2002.

[4] Altschul, S., and Erickson, B., "Optimal Sequence Alignment Using Affine Gap Cost," *Bulletin of Mathematical Biology*, vol. 48, pp. 603-616, 1986.

[5] Smith, T., and Waterman, M., "Identification of Common Molecular Subsequences," *Journal of Molecular Biology*, vol. 147, pp. 195-197, 1981.

[6] Gotoh O., "An Improved Algorithm for Matching Biological Sequences," *J Mol Biol.*, vol. 162, pp. 707-708, 1982.

[7] Tompa, M., *Basics of Molecular Biology*. Seattle: Department of Genome Science University of Washington, 2009.

[8] Liu, Y., Maskell D. L., and Schimdt, B., "CUDASW++: Optimizing Smith-Waterman Sequence Database Searches for CUDA-enabled Graphics Processing Units," *BMC Research Notes*, vol. 2, no. 73, 2009.

[9] Bustamam, A., K. Burrage, and N. A. Hamilton, "Fast Parallel Markov Clustering Using Massively Parallel Computing on GPU with CUDA and ELLPACK-R Sparse Format," *IEEE/ACM Transaction on Computational Biology and Bioinformatics*, vol. 9 no. 3, pp. 679-692, 2012.

[10] Browan, W. M., Wang P., Plimpton, S. J., and Tharrington, A. N., "Implementing molecular dynamics on hybrid high performance computers–short range forces," *Computer Physics Communications*, vol. 182, no. 4, pp. 898–911, 2011

[11] Lindholm, E., Nickolls, J., Oberman, S., & Montrym, J., "NVIDIA Tesla: A Unified Graphics and Computing Architecture," IEEE Computer Society. 28(2):39-55, 2008.

[12] Kirk, D. B., and W. W. Hwu. *Programming Massively Parallel Processors: A Hands-On Approach*, 1st ed., Morgan Kaufmann Publishers Inc., San Francisco, CA, 2010.

[13] NVIDIA. *CUDA C Programming Guide 5.0*. Santa Clara: NVIDIA Press, 2012.

[14] Nickolls, J., Buck, I., Garland, M., and Skadron, K., "Scalable Parallel Programming with CUDA," *ACM Queue*, vol. 6, no. 2, pp. 40-53, 2008.

[15] Liu, Y., Maskell, D. L., and Schimdt, B., "CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions," *BMC Research Notes*, vol. 3, no. 93, 2010.

[16] Henikoff, S., and Henikoff, J. G., "Amino acid substitution matrices from protein blocks," *Proc. Natl. Acad. Sci. USA*, vol. 89, no.22, pp. 10915-10919, 1992.

[17] Khajeh-Saeed, A., Poole, S., and Perot, B., "*Acceleration of the Smith-Waterman Algorithm Using Single and Multiple Graphic Processor*," *Journal of Computational Physics*, vol. 229, pp. 4247-4258, 2010.