Hybrid Framework for pairwise DNA Sequence Alignment Using the CUDA compatible GPU

H. Khaled, R. El Gohary, N.L. Badr and H. M. Faheem
Faculty of Computer & Information Science, Ain Shams University,
Cairo, Egypt.
heba.kaled@cis.asu.edu.eg, dr.raniaelgohary@fcis.asu.edu.eg, dr.nagwabadr@gmail.com,
hmfaheem@cis.asu.edu.eg

***Abstract*—**This paper provides a novel framework for accelerating the solution of the pairwise DNA sequence alignment problem using CUDA parallel paradigm available on the NVIDIA GPU. The main idea is to implement a new algorithm that assigns different nucleotide weights using GPU architectures then merge the subsequences of match using CPU to get the optimum local alignment. The paper describes both the algorithm and the implementation of it using both the GPU and CPU to constitute a hybrid model for solving DNA sequence alignment problem on DNA molecules. Experimental results demonstrate a considerable reduction in run time relative to traditional Smith-Waterman implementation on traditional processors.

***Keywords*—** GPU, GPGPU, CUDA, sequence alignment algorithms, molecular biology.

1 Introduction

Sequence comparison is a very basic and important operation in Bioinformatics as Sequence alignment is a key component in the analysis of genes and genomes. Sequence alignment algorithms find regions in one sequence, called the query sequence, that are similar or identical to regions in another sequence, called the reference sequence [1]. A sequence alignment has a similarity score associated to it obtained by placing one sequence above the other, making clear correspondence between the characters and possibly introducing gaps into them. The most common types of sequence alignment are global and local. To solve a global alignment problem it is required to find the best match between the entire sequences. On the other hand, local alignment algorithms must find the best match between parts of the sequences. Both Needleman-Wunsch algorithm [2] for global alignment and Smith-Waterman algorithm [3] for local alignment deploy dynamic programming approaches.

Genomic databases have an exponential growth rate. Therefore, a huge amount of new DNA sequences will need to be compared, in order to infer functional/structural characteristics. The growth of database size increases the time required for searching using this kind of dynamic programming approaches. Complexity of sequence comparison is proportional to query size and database size [4], [5].

The recent development of multi-core architectures, and its associated programming interfaces, provide an opportunity to accelerate sequence database searches using commonly available and inexpensive hardware.

Graphics hardware is currently deployed in high-performance computing due to its cost effectiveness. Bioinformatics applications also exploit GPU as a massive parallel multi-core processor to address computational challenges in many areas such as sequence analysis and alignment and protein structure prediction [6].

CUDA is the architecture and developing platform of the NVIDIA GPU. It is an extension of the C programming language. CUDA programs typically consist of a component that runs on the CPU, or host, and a smaller but computationally intensive component called the kernel that runs in parallel on the GPU. The kernel cannot access the CPU's main memory directly – input data for the kernel must be copied to the GPU's on-board memory prior to invoking the kernel, and output data also must first be written to the GPU's memory. All memory used by the kernel must be pre-allocated, and the kernel cannot use recursion or other features requiring a stack, but loops and conditionals are allowed [1].

In CUDA, the GPU is viewed as a computing device suitable for parallel data applications. It has its own device random access memory and may run a huge number of threads in parallel [7] as shown in Fig.1. Threads are grouped in blocks and many blocks may run in a grid of blocks. Such structured sets of threads could be launched on a kernel of code and process the data stored in the device memory. Threads of the same block share data through fast shared on chip memory and they can be synchronized through synchronization points as shown in Fig.2 [8], [9]. The proposed DNA sequence alignment approach can benefit from the CUDA architecture and the single instruction multiple thread SIMT model.

The SIMT completes the DNA sequence comparison in two stages; the first stage is used to find matches and mismatches between each nucleotide from both the query and the target sequences. The second stage is used to weight and highlight the subsequences of matches. The resulting subsequences of matches are then passed to the CPU to be merged in order to find the optimum alignment between the two sequences.

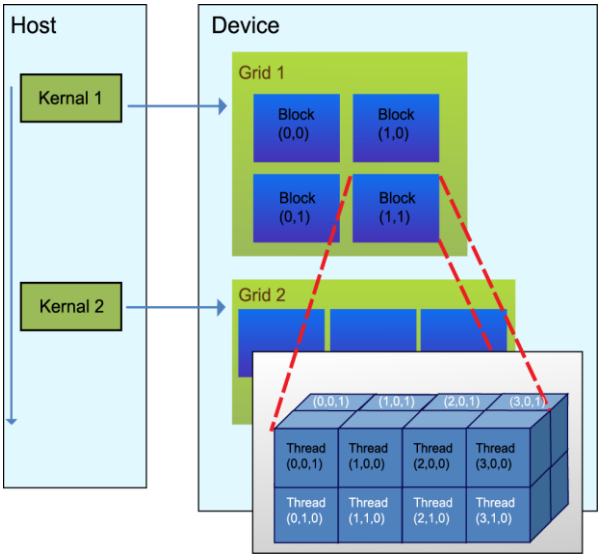

Fig. 1 Heterogeneous programming.

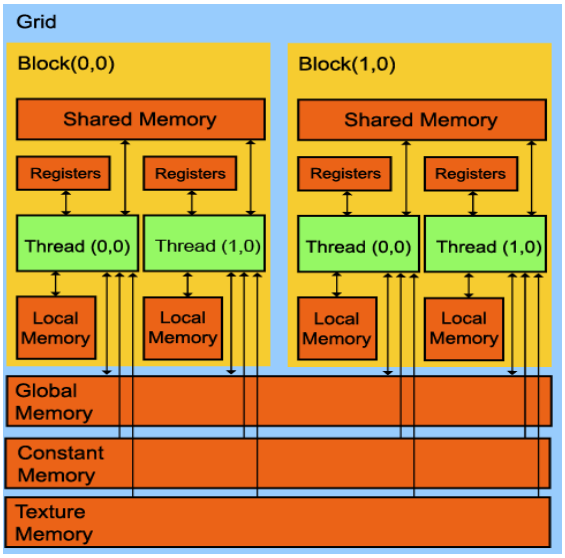

Fig. 2 CUDA Memory Model

The rest of the paper is organized as follows: Section 2 describes the proposed framework. Section 3 describes the operation of the proposed system Section 4 provides the experimental results. Section 5 augments some concluding remarks.

2 Proposed Framework

Task Dependency in Smith-Waterman algorithm is very high. For each cell in the Smith-Waterman dynamic programing matrix, we need to compute the upper, left and diagonal cells adjacent to that cell. This is to find the best alignment between two DNA sequences.

Given two DNA sequences S of length n, and T of length m, the proposed algorithm starts with an initialization phase shown in Fig.4. During this phase, a sufficient number of threads that can carry out n × m initialization operation is activated. Corresponding locations in the two sequences are compared such that a weight of 2 is granted to matched positions and 0 is granted to unmatched.

Given two DNA sequences S of length n, and T of length m:

1- Activate a number of threads that can carry out n × m initialization operation.
2- Load the two DNA sequences to all the threads.
3- For All the threads Perform the initialization process according to the following rules:

IF $S_i = T_J$ $i=1\ldots n.$ $j=1\ldots m.$

THEN $H(i, j)_t = 2$

ELSE IF $S_i \neq T_J$ where H_t is the preprocessing matrix

THEN $H(i, j)_t = 0$

Fig. 4 Initialization Algorithm

Sequence matching process is then performed as described in Fig.5. During this phase, a weight of 4 is granted to a specific position in the matrix having an initial weight of 2 if at least one of its adjacent upper diagonal left or lower diagonal right positions have a weight of 2. If both of the mentioned adjacent positions have a weight of 2 then, a weight of 6 is granted to the specific position. This approach maximizes the score of continuous "subsequence matching."

1- Activate a number of threads that can carry out n × m matching operation.
2- For All the threads Perform the sequence matching process according to the following rules:

IF $H(i, j)_t = 0$ where H_t and H_{t+1} are the

THEN $H(i, j)_{t+1} = 0$ preprocessing and the final sequence

ELSE IF $H(i, j)_t = 2$ matching matrixes

THEN

IF $H(i+1, j+1)_t = 2$ ***AND*** $H(i-1, j-1)_t = 2$

THEN $H(i, j)_{t+1} = 6$

ELSE IF $H(i-1, j-1)_t = 2$ ***OR*** $H(i+1, j+1)_t = 2$

THEN $H(i, j)_{t+1} = 4$

ELSE $H(i, j)_{t+1} = 2$

Fig. 5 Sequence Matching Algorithm

It is clear that there is no task dependency either in the initialization phase or in sequence matching process now, we are ready to implement this parallel part using CUDA provided by NVidia GPU which can lead to a significant improvement in the speed without the need to deploy special purpose hardware as in [10].

The block diagram shown in Fig.6 illustrates the model used to implement the Hybrid system for DNA sequence alignment using GPU.

The Implementation flow of the DNA Sequence Alignment consists of four subsystems: Initialization, pre-processing, Alignment, and the Output subsystem.

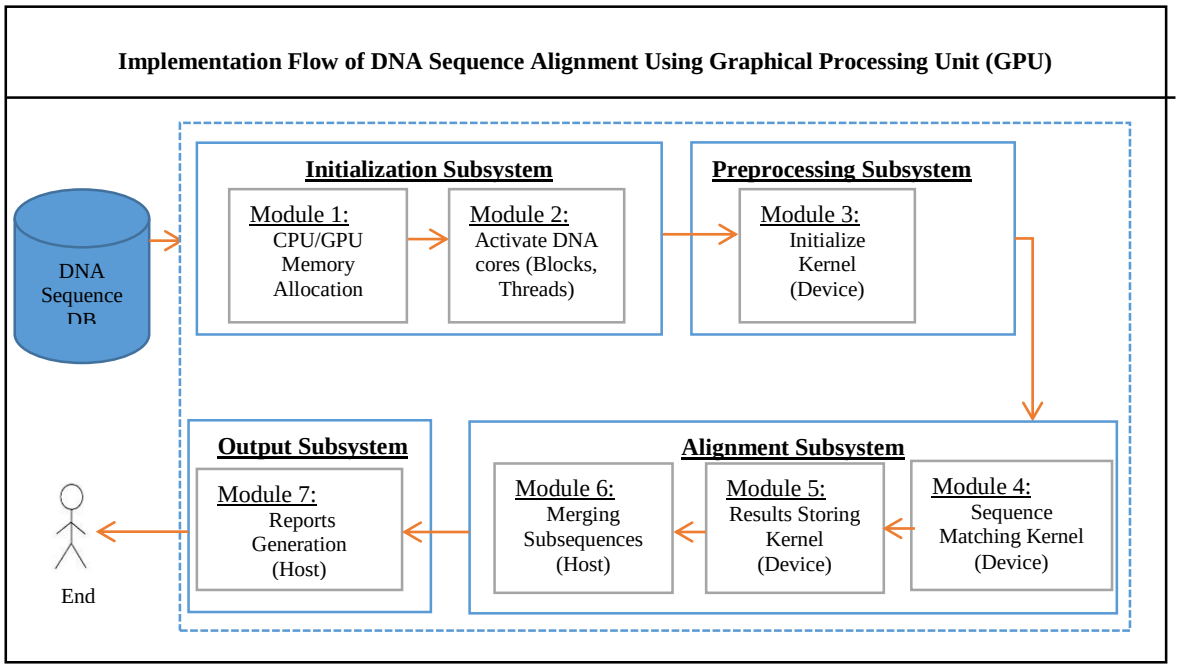

Fig. 6 Implementation Flow of DNA Sequence Alignment Using Graphical Processing Unit (GPU).

The Initialization Subsystem is concerned with the initialization phase that allocates memory for both CPU and GPU used to read the DNA sequences and store the output. It also activates number of CUDA Cores (blocks and threads) according to the sizes of the two DNA input sequences and then passes the inputs to the GPU grid.

At this point, the Preprocessing Subsystem can work on the device (GPU); the initialization kernel purpose is to compare each DNA nucleotide in the two sequences and fill the initialization vector with values for both DNA nucleotide match and mismatch. All the thread blocks perform the exact matching between the corresponding first sequence nucleotide and second sequence nucleotide simultaneously as a first step in the initialization process.

The resulting output matrix of the pre-processing subsystem is then passed to the Alignment subsystem.

The Alignment subsystem first applies the sequence matching algorithm that also works on the device (GPU). The sequence-matching kernel highlights the subsequence of match between the query and the subject sequences by weighting the match and the sub-sequences of match.

The proposed framework takes the advantage of the fact that each nucleotide comparison "initialization then Matching weight" for both the query sequence and the subject sequence can be computed independent of each other. Therefore, a number of n threads in a thread block are responsible for computing a row in the alignment matrix H.

Increasing the DNA sequences' size requires increasing the number of initialization and sequence matching operation needed. Using the CUDA architecture makes it obvious to scale the number of threads needed for the initialization and the sequence matching kernels.

The GPU based part is scalable as different number of blocks and threads per block are activated according to the given query and sequence sizes.

The resulting matrix from the Sequence matching kernel is then passed from the GPU to the CPU host memory where the last module "Merging Subsequences" in the Alignment subsystem will operate.

The Merging subsequences function running on the Host "CPU" firstly sorts the subsequences of match ascending according to their score. It represents the subsequence as entries in a table of indices and then tries to link these entries according to an input threshold given by the user until the optimum alignment could be found. The algorithm listed in Fig.7 represents how the Merging subsequences function operates.

The algorithm first creates a table of indices, which contains a list of matched subsequences represented by both lead and trail represented by i and j coordinates" of each matched subsequence, its score, and Mismatch/gap. A preprocessing step is then carried out to discard the very small subsequences of score equals to 2 that represents "only one match". The algorithm then sorts the entries of the matched subsequences using merge sort.

The user can specify a merging threshold K "the longest common subsequences between the two DNA sequences" that indicates how many subsequence to be merged with the whole entries in the table of indices. The algorithm then merges the entries in the table of indices "matched subsequences" according to a set of rules listed in the DNA sequence alignment sequential algorithm. The merging operation's complexity is O (K N) where K is a constant representing the merging threshold and N is the size of the table of indices. After the merging process the alignment of maximum score can be found.

1- Apply merge sort to the stored results, it sorts the subsequences descending according to their score "A preprocessing step".
2- Discard small subsequences of score equal 2.
3- Create table of indices according to the sorted subsequences.
4- Given that Each entry "aligned subsequence" in the table of indices has a lead and trail and each has i and j coordinates such that:

- i^{st}_{Lead} Is the i coordinate of the first subsequence's lead,
- j^{st}_{Lead} Is the j coordinate of the first subsequence's lead,
- i^{st}_{Trail} Is the i coordinate of the first subsequence's Trail,
- j^{st}_{Trail} Is the j coordinate of the first subsequence's Trail,
- i^{nd}_{Lead} Is the i coordinate of the second subsequence's lead,
- j^{nd}_{Lead} Is the j coordinate of the second subsequence's lead,
- i^{nd}_{Trail} Is the i coordinate of the second subsequence's Trail,
- j^{nd}_{Trail} Is the j coordinate of the second subsequence's Trail,
- $Score^{st\ Seq}$ = First subsequence's score,
- $Score^{nd\ Seq}$ = Second subsequence's score.
- $M\_G = \max[(i^{nd}_{Lead} - i^{st}_{Trail}), (j^{nd}_{Lead} - j^{st}_{Trail})]$ =Gaps and/or Mismatches between the two subsequences,
- Given a threshold K indicating how many subsequence to be merged with the whole entries in the table of indices, Merge the subsequences in the table of indices according to the following rules:

IF ((($i^{st}_{Trail} == i^{nd}_{Lead}$ AND $j^{st}_{Trail} < j^{nd}_{Lead}$)**OR**
($j^{st}_{Trail} == j^{nd}_{Lead}$ AND $i^{st}_{Trail} < i^{nd}_{Lead}$))
AND $(M\_G < Score^{st\ Seq})$ **AND** $(M\_G + 1 < Score^{st\ Seq})$)
THEN
Total score= $Score^{st\ Seq} + Score^{nd\ Seq} - M\_G - 2$

ELSE IF ((($i^{st}_{Trail} < i^{nd}_{Lead}$) **AND** ($j^{st}_{Trail} < j^{nd}_{Lead}$))
AND$((M\_G - 1) < Score^{st\ Seq})$
AND $((M\_G - 1) < Score^{ndSeq})$)
THEN
Total score=$Score^{st\ Seq} + Score^{nd\ Seq} - (M\_G - 1)$

5- Add the merged subsequences to the table.
6- After a round of merging and getting new subsequences, delete from the table of indices the first subsequence used in margining each new subsequence.
7- Go to 3 "another round of margining" until there are no sequences to be merged.

Figure7 DNA Sequence Alignment Sequential Algorithm

The Output Subsystem provides the results by selecting and reporting the sequence of maximum score and minimum gaps and mismatches.

To sum up, the proposed framework aims to compare nucleotides from both the query and the subject sequences, weight the matched subsequences, and then link the matching subsequences. Finally, the optimum local alignment will be the sequence of the highest score.

3 Operation

In order to explain the operation of the proposed system, let us consider an example; perform sequence alignment on the given two DNA sequences S= $T^1C^2G^3C^4A^5G^6A^7$ of length n=7 and T= $T^1C^2C^3A^4G^5C^6A^7$ of length m=7. The operation shown in Table 1 and Table 2 can be summarized as follows:

- Activate a total number of threads equals n*m =49 to perform both the initialization and the sequence matching processes. This can be carried out either by distributing these threads among number of blocks or activating only one block of total number of 49 threads. When activating the threads per block we consider the CUDA architecture limitation: a maximum of 512 threads per block and the maximum number of blocks per grid equals 65535.
- Load the two DNA sequences to the activated threads.
- All threads perform the initialization process "Kernel" simultaneously according to the rules stated in the Initialization Kernel. The initialization matrix will contain the values shown in Table 1.
- After the initialization process, all the threads perform the sequence matching process "kernel" simultaneously according to the rules stated in the Sequence Matching Kernel. The resulting matrix will be as shown in Table 2.

TABLE 1
Matrix Values after Initialization Kernel

	T	**C**	**G**	**C**	**A**	**G**	**A**
T	2	0	0	0	0	0	0
C	0	2	0	2	0	0	0
C	0	2	0	2	0	0	0
A	0	0	0	0	2	0	2
G	0	0	2	0	0	2	0
C	0	2	0	2	0	0	0
A	0	0	0	0	2	0	2

TABLE 2
Matrix Values after Sequence Matching Kernel

	T	**C**	**G**	**C**	**A**	**G**	**A**
T	4	0	0	0	0	0	0
C	0	4	0	2	0	0	0
C	0	2	0	4	0	0	0
A	0	0	0	0	6	0	2
G	0	0	4	0	0	4	0
C	0	2	0	6	0	0	0
A	0	0	0	0	4	0	2

- Pass the resulting matrix to the Host to fill the initial entries of the table of indices as shown in Table 3 according to the rules stated in the Merging subsequences function.
- Perform the preprocessing step to discard very small subsequences of score equals to 2 and sort the entries of the table of indices descending according to their score using merge sort as shown in Table 4.
- Merge the subsequences in the table of indices according to a threshold K indicating how many subsequences are used in merging with the whole entries of the table of indices. Add the merged subsequences to the table. The worst case in this sequential part occurs when the threshold K equals to the total number of entries in the table of indices the algorithm will try to merge each entry in the table of indices with the whole entries.
- Delete from the table of indices the first subsequence used in margining each new subsequence.

TABLE 3
Initial Table of Indices

ID	Subsequence &	Lead	Trail	Score	Gap& Mismatch
0	_	(2,4)	(4,6)	6	0
1	_	(3,2)	(5,4)	6	0
2	_	(0,0)	(1,1)	4	0
3	_	(1,2)	(1,2)	2	0
4	_	(1,5)	(1,5)	2	0
5	_	(3,1)	(3,1)	2	0
6	_	(6,3)	(6,3)	2	0
7	_	(6,6)	(6,6)	2	0

TABLE 4
Table of Indices after Discarding Very Small Subsequences

ID	Subsequence &	Lead	Trail	Score	Gap& Mismatch
0	_	(2,4)	(4,6)	6	0
1	_	(3,2)	(5,4)	6	0
2	_	(0,0)	(1,1)	4	0

- Repeat the merging operation consequently.
- If there is nothing to be merged then select the sequence of maximum score and minimum gaps and mismatches.
- The resulted tables of indices is shown in Table 5 given threshold K=3.

TABLE 5
Complete Round in the Table of Indices at K=3

ID	Subsequence &	Lead	Trail	Score	Gap& Mismatch
0	2&1	(0,0)	(5,4)	9	1
1	2&0	(0,0)	(4,6)	8	2
2	0	(2,4)	(4,6)	6	0
3	1	(3,2)	(5,4)	6	0

- The best alignment starts at lead (0, 0) and ends with the trail (5,4) with score equals 9 and Gap and Mismatch equals 1 which is indicated at sequence number 0 at the final round in the table of indices.

4 Performance Evaluation

The performance of the proposed algorithm is measured by comparing the execution time of the NVidia GPU/CPU version of the preprocessing, sequence matching and subsequences merging running time of the DNA sequence alignment proposed algorithm versus both the sequential execution of the same proposed algorithm of sequence comparison [10], and the sequential Smith-Waterman algorithm [3] used for DNA sequence alignment implemented on the same machine.

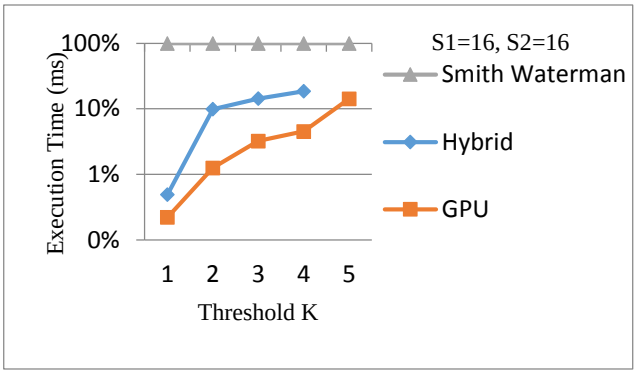

Fig. 8 Hybrid system and Smith-Waterman execution times at S1=16 and S2=16 BP

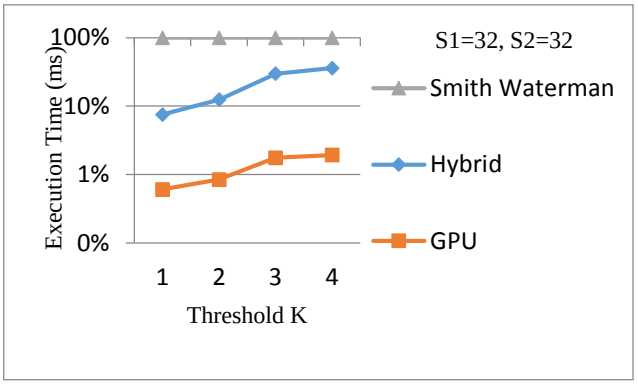

Fig. 9 Hybrid system and Smith-Waterman execution times at S1=32 and S2=32 BP

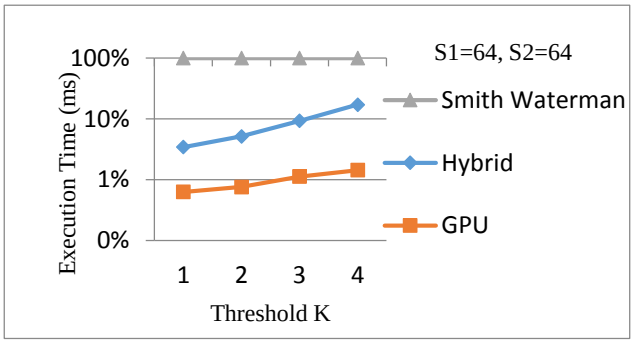

Fig. 10 Hybrid system and Smith-Waterman execution times at S1=64 and S2=64 BP

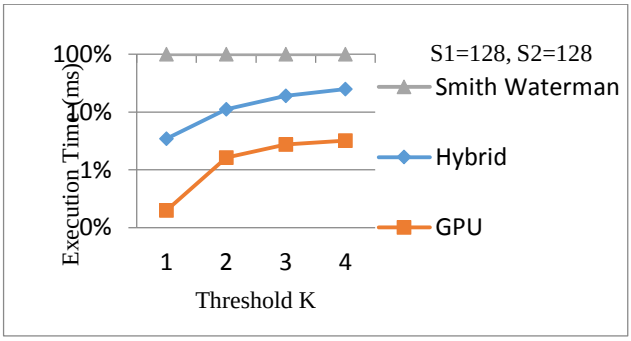

Figure 11 Hybrid system and Smith-Waterman execution times at S1=128 and S2=128 BP

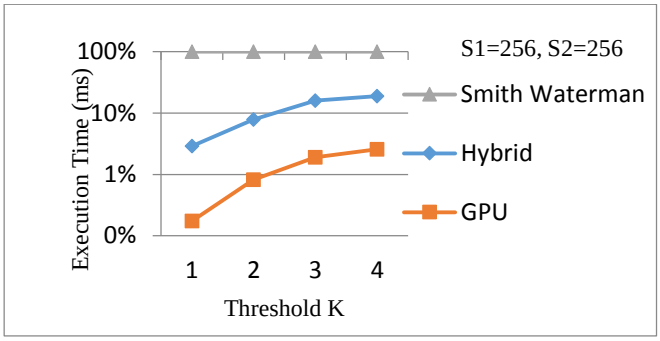

Figure 12 Hybrid system and Smith-Waterman execution times at S1=256 and S2=256 BP

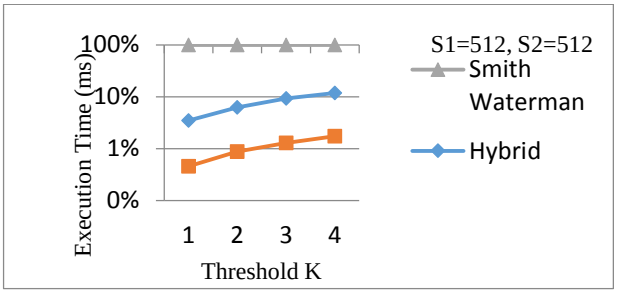

Figure 13 Hybrid system and Smith-Waterman execution times at S1=512 and S2=512 BP

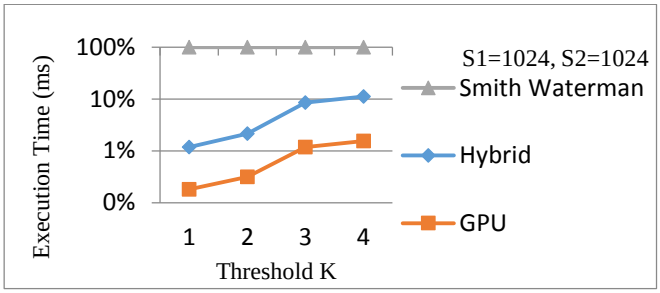

Figure 14 Hybrid system and Smith-Waterman execution times at S1=1024 and S2=1024 BP

Table 6
The Proposed GPU Implementation, The Hybrid System and Smith-Waterman Execution Times for Different Sequence Sizes.

Sequence's Size		Total Hybrid System Time (Parallel Matching GPU +Sequential Rounds) (ms)					Total Hybrid System "parallel + Sequential Execution "(ms)					Smith Waterman
S1	S2	T at K=1	T at K=2	T at K=3	T at K=4	T at K=5	T at K=1	T at K=2	T at K=3	T at K=4	T at K=5	
16	16	0.1389	0.3571	0.3277	0.4233	0.9589	0.0141	0.4982	0.6718	0.8997	3.2393	5.19781
32	32	0.4010	0.4964	1.4243	1.0972	1.6248	1.3449	2.3729	7.1229	9.5471	6.1988	17.983
64	64	1.3591	1.3024	1.8715	2.5768	4.2268	2.1330	3.3937	6.6120	13.9064	20.1797	73.4924
128	128	2.3459	14.8844	15.9078	19.8200	12.8086	9.8944	31.1223	58.8689	83.9278	85.9526	291.548
256	256	4.6377	16.6957	58.8322	106.7312	57.8178	29.8864	81.0289	177.6650	212.0680	256.1450	1059.95
512	512	16.7758	33.3471	50.2516	66.9213	386.2223	106.7730	194.8590	299.6170	390.4650	2328.3000	3397.86
1024	1024	41.3386	101.2055	282.8225	305.7553	316.2523	138.7090	260.3110	1115.4600	1488.2800	1852.3400	13817.8

The list of figures from Fig.8 to Fig.14 shows the Hybrid system execution time using GPU, Parallel architecture in [10] and Smith-Waterman at different sequence sizes starting from 16 bp 'Base pair' to 1024 bp.

The GPU execution time for both preprocessing and sequence matching is recorded at different input sequences' size and the CPU execution time for the merging subsequences is calculated for different thresholds as shown in Table 6.

Results are obtained using Intel Core i5 2430M 2.4GHZ, 4 GB DDR3 Memory and NVidia GeForce GT540M GPU with

96 CUDA Cores with 1GB device memory. All the implementations run on Windows7 with Display Driver285.86.

The methods are implemented using Microsoft Visual Studio 2010 and NVidia GPU Computing SDK 4.1.

5 Conclusion

The proposed framework combines both a parallel and a sequential algorithm to speed up the solution of the pairwise DNA sequence alignment. The architecture of the hybrid system uses the GPGPUs. It has been observed that the proposed framework can provide an alignment quality comparable to that of Smith-Waterman algorithm while consuming significantly less time.

The target of the proposed framework is to compare all the nucleotides from both the query and the target sequences simultaneously then extract the subsequences of match and try to merge them to find the optimum alignment according to the maximum score and minimum gap/mismatch.

The system is considered a step towards a complete parallel processing architecture to solve computationally intensive applications of DNA

6 References

[1] Michael Schatz, Cole Trapnell, Arthur Delcher, Amitabh Varshney, *"High-throughput sequence alignment using Graphics Processing Units,"* BMC Bioinformatics, Vol. 8, No. 1, 2007.

[2] T. F. Smith and M. S. Waterman, *"Identification of common molecular subsequences,"* J Mol Biol, 147(1), pp. 195-197, March 1981.

[3] T. Smith and M. Waterman, *"Identification of common molecular subsequences,"* J. Mol. Bio., (147):195–197, 1981.

[4] J. Setubal and J. Meidanis, *Introduction to Computational Molecular Biology*, PWS Publishing Company, 1997.

[5] Terence Hwa and Michael Lässig, *"Similarity Detection and Localization,"* Physical Review Letters Volume: 76, Issue: 2, 1995.

[6] Jun Sung Yoon and Won-Hyong Chung, *"A GPU-accelerated bioinformatics application for large-scale protein interaction networks,"* Asia Pacific Bioinformatics Conference, 2011.

[7] Rafia Inam, *"An Introduction to GPGPU Programming - CUDA Architecture,"* Mälardalen University, Mälardalen Real-Time Research Centre, 2011.

[8] NVIDIA CORPORATION, CUDA Programming Guide, *http://developer.nvidia.com/category/zone/cuda-zone*

[9] Svetlin A Manavski and Giorgio Valle1, *"CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment,"* BMC Bioinformatics 2008.

[10] Heba Khaled , Hossam M Faheem , Tayseer Hasan , Saeed Ghoneimy,*"Design of a Hybrid System for DNA Sequence Alignment,"* Proceedings of The International MultiConference of Engineers and Computer Scientists 2008 , pp162-167.