Accelerating Pairwise DNA Sequence Alignment using the CUDA Compatible GPU

H. Khaled
Faculty of Computer & Information Science, Ain Shams University, Cairo, Egypt

R. El Gohary
Faculty of Computer & Information Science, Ain Shams University, Cairo, Egypt

N.L. Badr
Faculty of Computer & Information Science, Ain Shams University, Cairo, Egypt

H. M. Faheem
Faculty of Computer & Information Science, Ain Shams University, Cairo, Egypt

ABSTRACT
We present a novel implementation of the pairwise DNA sequence alignment problem other than the Dynamic programming solution presented by Smith Waterman Algorithm. The proposed implementation uses CUDA; the parallel computing platform and programming model invented by NVIDIA. The main idea of the proposed implementation is assigning different nucleotide weights then merging the sub-sequences of match using the GPU Architecture according to predefined rules to get the optimum local alignment. We parallelize the whole solution for the pairwise DNA sequence alignment using CUDA and compare the results against a similar semi-parallelized solution and a traditional Smith-Waterman implementation on traditional processors; Experimental results demonstrate a considerable reduction in the running time.

General Terms
Bioinformatics, HPC, Parallel Processing.

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

1. INTRODUCTION
Sequence comparison is a very basic and important operation in Bioinformatics. Sequence alignment algorithms detect similar or identical parts between two sequences called the query sequence and the reference sequence [1]. The global and local sequence alignment are the most prevalent kinds of sequence alignment. In global alignment problem finds the superior counterpart between the whole sequences. On the other hand, local alignment algorithms must find the superior counterpart between parts of the sequences.

Genomic databases have an exponential growth rate. The growth of database size increases the time required for searching. Complexity of sequence comparison is proportional to query size and database size [2], [3].

The recent development of multi-core architectures provide an opportunity to accelerate sequence database searches using available and inexpensive hardware.

CUDA is the architecture and developing platform of the NVIDIA GPU. It is a C like programming language. CUDA programs contains code that run on both the CPU, or host, and the GPU or as called device. The Kernel is the code running on the GPU; it contains the computationally intensive parts of the program [4].

GPU is the computing device suitable for parallel data applications. GPU has its own device random access memory and may run a huge number of threads in parallel [5] as shown in Figure.1. Blocks consists of threads, and many blocks can run within a grid of blocks. This structured sets of threads could be launched on a kernel of code and process the data stored in the device memory. Figure.2 shows that threads of the same block share data through fast shared on chip memory and they can be synchronized through synchronization[6], [7]. The proposed implementation benefits from the CUDA architecture and the single instruction multiple thread SIMT model.

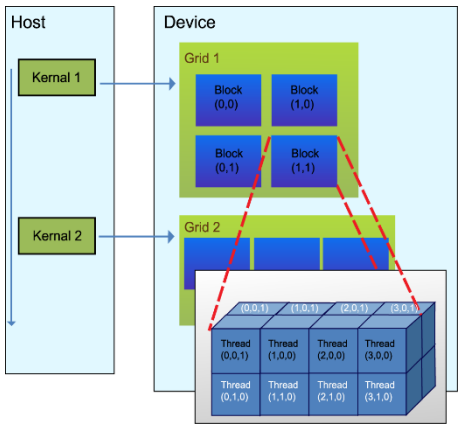

Fig. 1 Heterogeneous programming [5]

In the proposed implementation, the SIMT completes the DNA sequence comparison in three stages; the first stage finds matches and mismatches between each nucleotide from both the query and the target sequences. The second stage weights and highlights the subsequences of matches. The third stage merges the resulting subsequences of matches in order to find the optimum alignment between the two sequences.

We organize the rest of the paper as follows: Section 2 describes the related work. Section 3 describes the proposed framework. Section 4 provides the experimental results. Section 5 augments some concluding remarks.

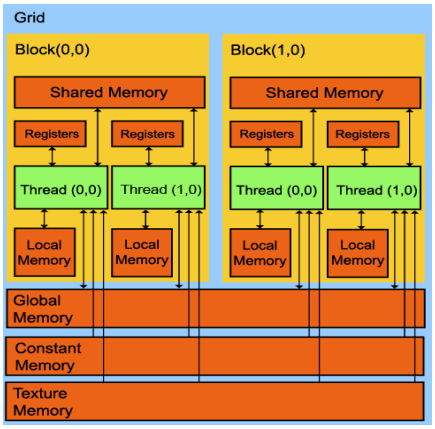

Fig. 2 CUDA Memory Model [6][7]

2. RELATED WORK

There are many approaches used to solve sequence alignment problem. Dynamic programming, heuristic approach, and linear space alignment are of those approaches. Tools using the heuristic approach like FASTA [8] and BLAST [9] are practical as they find the near optimal solution; they are approximation algorithms that cannot give the optimal solution. Both Needleman-Wunsch algorithm [10] for global alignment and Smith-Waterman algorithm [11] for local alignment use dynamic programming approaches. However, the complexity of the dynamic programming approach is O (MN), where M and N represent the lengths of two DNA sequences been compared.

The Smith-Waterman algorithm is a variant of the Needleman-Wunsch. It has the same maximization step with the added value of 0. Let X = (x1, x2. . . xn) and Y = (y1, y2, . . , ym) be two sequences that are to be compared. Let d(x, y) be the substitution cost for changing x into y and g the cost of the insertion/deletion (gap) operation. H (i, j) is defined as the maximum similarity of two segments ending at xi and yj. The following recursion gives the Smith and Waterman algorithm.

$$H(i,j) = \max \begin{cases} 0 \\ H(i, j-1) - g \\ H(i-1, j-1) + d(xi, yj) \\ H(i-1, j) - g \\ H(i, 0) = H(0, j) = 0. \end{cases} \quad (1)$$

In Smith-Waterman algorithm for each cell in the dynamic programing matrix, we need to compute the upper, left and diagonal cells adjacent to that cell to find the best alignment between two DNA sequences.

Parallel processing and architecture was a very good solution for solving the DNA sequence alignment problems. Many implementations was done using the Nvidia GPUs to accelerate the Smith-Waterman algorithm.

Liu did the early implementation of Smith-Waterman using OpenGL-based implementation, reported in [12]. A parameterisable implementation on CUDA-compatible GPU's [13], this implementation used the divide and conquer approach to compute the alignment matrix by dividing the entire matrix computation into sub matrices and allocating available amount of threads and memory to each sub-matrix. That approach could achieve 4.2 GCUPS against Swiss-Prot database on Geforce 8800GTX and it was 15 times faster than the CPU implementation. A new parallel method for Smith-Waterman algorithm was introduced in [14]. The new method compute all the elements in the same column of the Smith-waterman Dynamic programming matrix independent of each other in parallel rather than computing all elements in the same anti-diagonal independently of each other in parallel. It exploits parallelization of the columns. That approach achieved 37 times speed up over the OSEARCH, but it fixed the target sequence length to 361base pair and used 176469 protein sequence.

The previous solutions was among many proposed Smith-Waterman solutions implemented for multiple sequence alignment. We present another novel implementation using the Nvidia CUDA capable GPUs but for the pairwise DNA sequence alignment.

3. METHODS

The proposed method for DNA sequence alignment using CUDA compatible GPUs consists of three main phases for the two DNA sequences S of length n, and T of length m to be alignment. The three phases are the initialization phase, the preprocessing phase and the alignment phase.

The Initialize Phase works at the Host side (CPU). It allocates memory in both CPU and GPU for passing the two DNA sequences and storing the alignment results. The initialization phase also reserves a number of CUDA blocks and threads per block to carry out the alignment operation; the number of activated blocks and threads depends on the size of the two DNA sequences.

The second phase is the Pre-processing Phase carried out on the device GPU. It compares each DNA nucleotide in the two sequences and fill the initialization vector at the GPU with values for both DNA nucleotide match and mismatch. At the pre-processing phase, we activate a sufficient number of threads that can carry out n × m initialization operation. All the thread blocks perform an exact matching between the corresponding first sequence nucleotide and second sequence nucleotide simultaneously. 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. Figure. 3 shows the pseudo code for both the initialization and preprocessing phases to align the two DNA sequences S and T of lengths n and correspondingly.

The third and final phase is the Alignment Phase executed on the device GPU; it works on the resulting output matrix of the pre-processing Phase. We divide the alignment phase to two main processes; the sequence matching process and the merging sub-sequences process.

The main purpose of the sequence-matching process is to highlight the subsequence of match between the query and the subject sequences by weighting the match and the sub-sequences of match. In the sequence matching process, a weight of 4 is granted to any 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" as shown in Fig4. The sequence-matching process then passes the resulting matrix to the CPU host memory to prepare it for merging.

```
/*****************Assumption***************/
Query Sequence S of length-> n
Subject Sequence T of length ->m
Resulting matrix from the pre-processing ->  devMatrix
*****************************************
1- Allocate Device memory for n and m.
2- Send S and T to the GPU.
3- Invoke the pre-processing Kernel to hold a max. N= n*m  comparison.
4- For All the threads Perform the pre-processing phase according to the following rules:
  tid= threadIdx.x+ blockIdx.x*blockDim.x;
  While ( tid<N)
    Get neculidiede from S and T where
       i=tid % n; //    i=1… n.
       j=tid/ n; // j=1… m.
    if(S[i]==T[j])
       devMatrix[tid]=2;
    else
       devMatrix [tid]=0;
    end if
       tid+=blockDim.x*gridDim.x;
end while
```

Fig. 3 Initialization and Pre-processing phases.

The Merging sub-sequences process is concerned with the final step of the proposed system that gets the final alignment of the given two sequences. We clearly defined the resulting matrix from the previous process "sequence-matching" by representing the sub-sequences of match by filling an index table that 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.

Using the index table located on the CPU host memory, we first apply a merge sort to sort all the index table entries ascending according to their score, and then discard the very small sub-sequences of score equals to 2 that represents "only one match" for being useless after getting all the huge sub-sequences of match. The resulting index table is then passed to the device GPU for the last processing step in the merging process.

The merging process tries to link the index table entries according to an input threshold given by the user until the optimum alignment could be found. Figure 5 represents how the Merging sub-sequences process operates.

We input a merging threshold K "the longest common sub-sequences between the two DNA sequences" that indicates how many subsequence to be merged with the whole entries in the table of indices. Given that the first sub-sequence is represented by its coordinates (i^{st}_{Lead} ,j^{st}_{Lead}), (i^{st}_{Trail} ,j^{st}_{Trail}) and its score is $Score^{st\ Seq}$ and the second sub-sequence is represented by its coordinates (i^{nd}_{Lead}, j^{nd}_{Lead}) and (i^{nd}_{Trail}, j^{nd}_{Trail}) and its score is $Score^{nd\ Seq}$ and the Gaps and/or Mismatches between the two sub-sequences is represented by equation (2).

$$M\_G = max[(i^{nd}_{Lead} - i^{st}_{Trail}), (j^{nd}_{Lead} - j^{st}_{Trail})] \qquad (2)$$

We perform the merging of the index table entries according to one of the following rules:

```
/*****************Assumption***************/
Input: devMatrix from the pre-processing phase.
Output:  ResultMatrix containing the sequence matching process results.
********** Sequence Matching kernel **********
1- Invoke the sequence matching Kernel to hold a max. N= n*m  comparison.
2- For All the threads Perform the sequence matching process and store results in the  ResultMatrix  according to the following rules:
tid= threadIdx.x+ blockIdx.x*blockDim.x;

While ( tid<N)
        Get neculidiede from S and T where
        i=tid % n; //    i=1… n.
        j=tid/ n; // j=1… m.
        updiag= tid - (n+1);
        downdiag= tid + n +1;
if(devMatrix [tid]==0)ResultMatrix[tid]=0;
else if (devMatrix [tid]!=0)
        if(((i==0) AND (j==m-1))OR((i==n-1) AND
        (j==0))) //top left & bottom right
                ResultMatrix [tid] = 2;
        else if((i==0)OR(j==0)) // up & left
                if(devMatrix[downdiag]==0)
                        ResultMatrix [tid]=2;
                else
                        ResultMatrix [tid]=4;
                end if

         else if((i==n-1)OR (j==m-1))//down&right
                if(devMatrix[updiag]==0)
                        ResultMatrix [tid]=2;
                else
                        ResultMatrix [tid]=4;
                end if
        else
                if((devMatrix[updiag]==0)AND
                (devMatrix[downdiag]==0))
                        ResultMatrix [tid]=2;
                else if((devMatrix[updiag]!=0 AND
                devMatrix [downdiag]==0)
                OR(devMatrix [updiag]==0 AND
                devMatrix [downdiag]!=0))
                        ResultMatrix [tid]=4;
                else if((devMatrix [updiag]!=0 AND
                devMatrix [downdiag]!=0))
                        ResultMatrix [tid]=6;
                end if
        end if
end if
tid+=blockDim.x*gridDim.x;
end While
```

Fig. 4 Sequence Matching Process

Rule 1: If the first sub-sequence's trail and the second sub-sequence's lead have the same i or j coordinate and the first sub-sequence proceeds the second one and the gab/mismatch between the two sub-sequences is less than the first sub-sequence's score, then the total score of the merge is shown in equation (3)

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** $(M\_G + 1 < Score^{st\ Seq})$)

$$Total\ score = Score^{st\ Seq} + Score^{nd\ Seq} - M\_G - 2 \quad (3)$$

Rule 2: If the first sub-sequence proceeds (up and left) the second sub-sequence and the gab/mismatch between the two sub-sequences is less than the first and the second sub sequences' scores, then the total score of the merge is shown in equation (4).

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

$$Total\ score = Score^{st\ Seq} + Score^{nd\ Seq} - (M\_G - 1) \quad (4)$$

```
/****************** *************** CPU Side ************************************/
1- Convert the ResultMatrix from the GPU resulting from the Sequence Matching process to an index Table.
2- Apply merge sort to the Index Table, and sorts the sub-sequences descending according to their score.
3- Discard small sub-sequences of score equal 2.
4- Pass the Index Table to the Merging Round Kernel.
/*********************************GPU Side*************************************/
Index Table Size→ ind_size.
Threshold K→ k_size.
Input: Index Table representing each sub-sequence with a lead and trail and each has i and j coordinates.
Output: Modified Index Table with the new merged sub-sequences → Mod_Ind_table.
/********************************* Merging Sub-sequences Kernel ********************************/
1- tid=  threadIdx.x+ blockIdx.x*blockDim.x;
2- The maximum number of merging per round is M=  ind_size*k_size;
while(tid<M)
        i=tid/ ind_size; //the k index
        j=tid % ind_size; //the size index
        Sequence seq1 =Index_Table[i];
        Sequence seq2= Index_Table[j];
  if(i!=j)
        Sequence seq3;
        m_g = max(((seq2.lead.i)-(seq.trail.i)),((seq2.lead.j)-(seq.trail.j)));
        if Rule (1) Applied
                1- Fill sequence 3 so that its (i,j) lead is equal to Sequence1 and its (i,j)trail is
equal to Sequence2.
                2- seq3.score = seq.score + seq2.score - m_g - 2;
                3- seq3.m_g = seq.m_g + seq2.m_g + m_g;
                4- Mod_Ind_table[tid]=seq3;
        else if Rule (2) Applied
                1- Fill sequence 3 so that its (i,j)lead is equal to Sequence1 and its (i,j)trail is equal
to Sequence2.
                2- seq3.score =seq.score + seq2.score - (m_g - 1);
                3- seq3.m_g = seq.m_g + seq2.m_g + m_g - 1;
                4- Mod_Ind_table[tid]=seq3;
        end if
  end if
tid+=blockDim.x*gridDim.x;
end while
3- After a round of merging and getting new sub-sequences, delete from the table of indices the first subsequence
used in margining each new subsequence.
4- Repeat the merging until there are no sequences to be merged.
```

Fig. 5 Merging Sub-Sequences Process.

The above rules and equations show that there is no task dependency for merging the index table entries, so we could execute this part efficiently on the device GPU.

We then add the new merged sub-sequences to the index table delete from the table of indices the first subsequence used in margining each new subsequence. We repeat the merging rules until there are no sequences to be merged.

By completing the merging process, we can point to the alignment of maximum score and minimum gaps and mismatches.

Increasing the DNA sequences' size requires increasing the number of pre-processing and sequence-matching operation, also increasing the threshold K leads to an increase in the number of rounds required to get the optimum alignment. Using the CUDA architecture makes it obvious to scale the number of threads needed for pre-processing, sequence matching and merging sub-sequences kernels.The GPU implementation is scalable as we can activate different number of blocks and threads per block according to the given query and target sequence sizes and the threshold K.

It is clear that there is no task dependency in any of the proposed three main phases the initialization, pre-processing and Merging Sub-sequences phases. Therefore, we implemented them 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 [15].

1. RESULTS

In this section, we present the experimental results of the proposed pairwise method for DNA sequence alignment implemented on GPU compared to another two similar approaches presented in both [15] and [16]. We implemented the proposed method using Microsoft Visual Studio 2010 and NVidia GPU Computing SDK 4.2. We used an 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.

Table 1 shows the total execution time for the three phases of the proposed method "Initialization, Pre-processing and Alignment phases" recorded at different input sequences' size starting from 16 bp 'Base pair' to 1024 bp and different thresholds K. It also shows the execution time of the similar Hybrid system [16] in which both the pre-processing and sequence alignment are executed on the GPU but all the merging rounds are done sequentially using the host CPU. The last section of Table 1 shows the execution time of the core functions in the proposed method but using special purpose hardware as in [15].

All the figures from Fig. 6 to Fig 11 emphasise on the big difference of the execution time for the proposed method, the Hybrid framework [16] and the special purpose HW approach [15] at different sequence sizes starting from 16 bp 'Base pair' to 1024 bp and for different thresholds K from 1 to 5.

The proposed method focussed on the pairwise DNA sequence alignment not the multiple sequence alignment. For that reason, we could not compare the results to the related work done using the Nvidia GPUs for multiple sequence alignment.

Table 1 The Proposed GPU Implementation, The Hybrid System and Special Purpose HW Execution Times for Different Sequence Sizes and Thresholds.

Sequence's Size		Parallel System Time (ms)					Hybrid System Time (Parallel Matching+Sequential Rounds) (ms) [16]					special purpose hardware Execution time(ms) [15]				
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	T at K=1	T at K=2	T at K=3	T at K=4	T at K=5
16	16	0.015	0.029	0.029	0.030	0.060	0.139	0.357	0.328	0.423	0.959	0.014	0.498	0.672	0.900	3.239
32	32	0.047	0.046	0.079	0.080	0.050	0.401	0.496	1.424	1.097	1.625	1.345	2.373	7.123	9.547	6.199
64	64	0.091	0.074	0.078	0.080	0.116	1.359	1.302	1.871	2.577	4.227	2.133	3.394	6.612	13.906	20.180
128	128	0.151	0.249	0.426	0.496	0.498	2.346	14.884	15.908	19.820	12.809	9.894	31.122	58.869	83.928	85.953
256	256	0.515	0.859	1.464	1.756	2.004	4.638	16.696	58.832	106.731	57.818	29.886	81.029	177.665	212.068	256.145
512	512	2.381	2.976	3.514	4.163	14.907	16.776	33.347	50.252	66.921	386.222	106.773	194.859	299.617	390.465	2328.300
1024	1024	7.604	8.356	12.711	15.465	16.677	41.339	101.205	282.823	305.755	316.252	138.709	260.311	1115.460	1488.280	1852.340

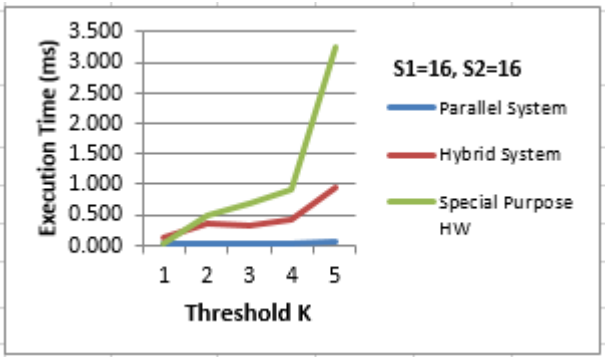

Fig. 6 Proposed method, the Hybrid framework and the special purpose HW approach execution times at S1=16 and S2=16 BP

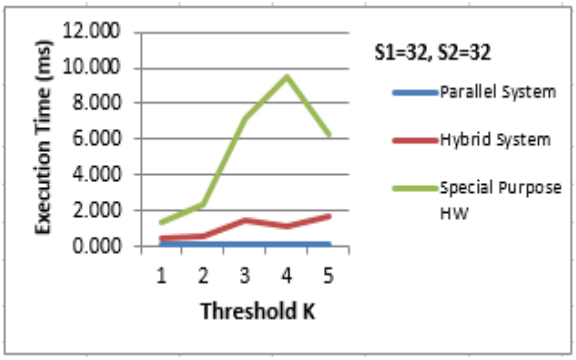

Fig. 7 Proposed method, the Hybrid framework and the special purpose HW approach execution times at S1=32 and S2=32 BP

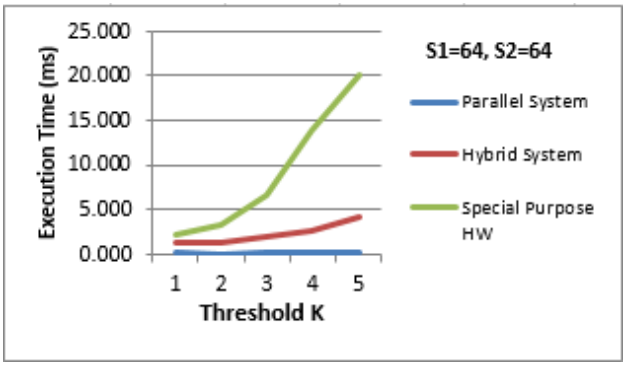

Fig. 8 Proposed method, the Hybrid framework and the special purpose HW approach execution times at S1=64 and S2=64 BP

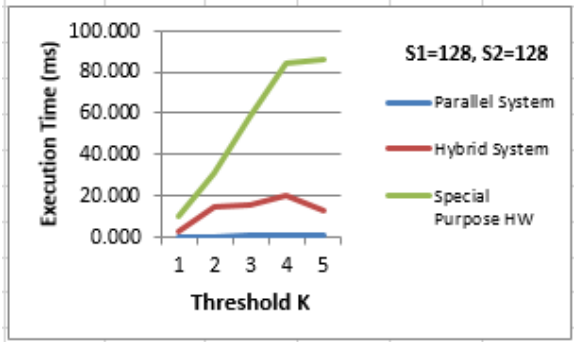

Fig. 9 Proposed method, the Hybrid framework and the special purpose HW approach execution times at S1=128 and S2=128 BP

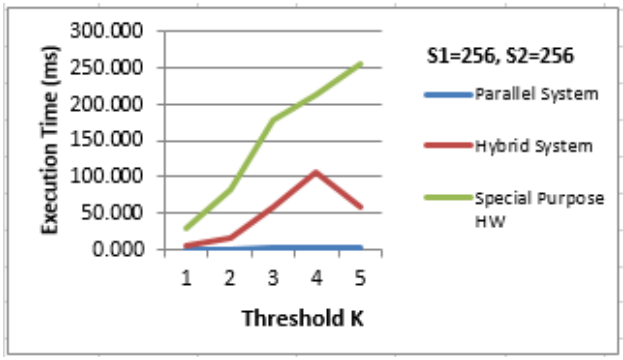

Fig. 10 Proposed method, the Hybrid framework and the special purpose HW approach execution times at S1=256 and S2=256 BP

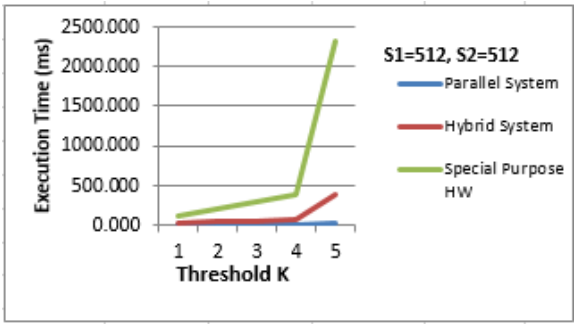

Fig. 11 Proposed method, the Hybrid framework and the special purpose HW approach execution times at S1=512 and S2=512 BP

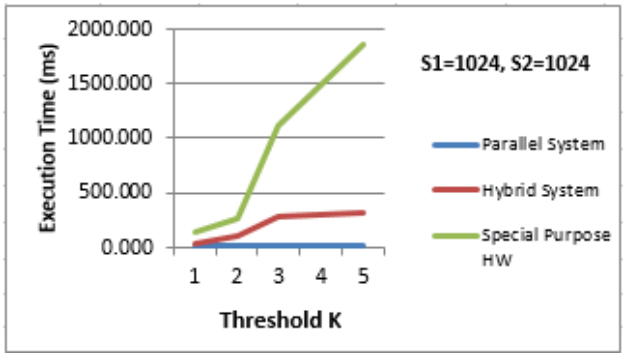

Fig. 12 Proposed method, the Hybrid framework and the special purpose HW approach execution times at S1=1024 and S2=1024 BP

2. CONCLUSION

The proposed Method accelerates the pairwise DNA sequence alignment by making use of the GPGPUs architecture. We compared the proposed method with both Smith-Waterman algorithm and the Hybrid system presented by [16], the proposed method shows an alignment quality while consuming significantly less time.

The proposed method compares the nucleotides of both the query and the target sequences simultaneously, it finds and weights the sub-sequences of match between the two sequences; it then merges the sub-sequences of match to get the final optimum alignment of the maximum score and minimum gap/mismatch.

The three main phases of the proposed system run on the device GPU and we pass the intermediate results to the host CPU.

3. REFERENCES

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

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

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

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

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

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

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

[8] Pearson, W.R. 1991. Searching protein sequence libraries: comparison of the sensitivity and selectivity of the Smith and Waterman and FASTA algorithms, Genomics 11, 635–650.

[9] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J., 1990. Basic local alignment search tool," J. Mol. Biol. 215, 403–410.

[10] S. Needleman and C.Wunsch, 1970 . A general method applicable to the search for similarities in the amino acid sequence of two proteins, J. Mol. Bio., (48):443–453.

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

[12] W. Liu, B. Schmidt, G. Voss, A. Schroder and W. Muller-Wittig, 2006. Bio-Sequence Database Scanning on GPU, In proceeding of 20th IEEE International parallel & distributed processing symposium, HICOMB workshop Rhode Island, Greece.

[13] Cheng Ling, Khaled Benkrid and Tsuyoshi Hamada, 2009. A Parameterisable and Scalable Smith-Waterman Algorithm Implementation on CUDA-compatible GPUs IEEE 7th Symposium on Application Specific Processors (SASP).

[14] Bo Chen, Yun Xu, Jiaoyun Yang, and Haitao Jiang, 2010. A New Parallel Method of Smith-Waterman Algorithm on a Heterogeneous Platform, Lecture Notes in Computer Science Volume 6081, pp 79-90, Springer-Verlag Berlin Heidelberg.

[15] 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.

[16] H. Khaled, R. El Gohary, N.L. Badr and H. M. Faheem, 2013. Hybrid Framework for pairwise DNA Sequence Alignment Using the CUDA compatible GPU, Proceeding of the BIOCOMP'13.

[17]