© Universiti Tun Hussein Onn Malaysia Publisher's Office

IJIE

Journal homepage: http://penerbit.uthm.edu.my/ojs/index.php/ijie
ISSN : 2229-838X e-ISSN : 2600-7916

The International Journal of Integrated Engineering

High-throughput Protein Sequence Alignment on Multi-core Systems

Muhammad Yahya[1,*], Laiq Hasan[2], Syed Asad Ali[3]

[1]University of Engineering and Technology Peshawar, UET, Peshawar, 25000, PAKISTAN

[2]University of Engineering and Technology Peshawar, UET, Peshawar, 25000, PAKISTAN

[3]CECOS University Peshawar, CECOS, Peshawar, 25000, PAKISTAN

*Corresponding Author

DOI: https://doi.org/10.30880/ijie.2020.12.07.007
Received 19 February 2020; Accepted 3 August 2020; Available online 30 August 2020

Abstract: Rapid evolution in sequencing technologies results in generating data on an enormous scale. A focal and main challenge in analyzing data at such a large scale is the alignment of the DNA/Protein sequences, whereby reads are compared to the reference sequences. To find similar sequences, alignment algorithms are used to align a query sequence with the database. Alignment algorithms can be utilized to classify the source of a sequence, to discover similarities among the organisms, or to deduce a progenitor connection. A wide range of algorithms for alignment has been developed in recent years. In this paper, an accurate method of accelerating such algorithms using GPUs has been investigated. A Swiss-Prot database has been processed using GPU implemented Smith-Waterman Sequence Alignment Algorithm. The first step in the process generates the alignment scores but not the actual alignment. Various available alignment tools like ssearch2 are then utilized to align the output file generated during the first step. The performance of GPU-accelerated implementation as compared to other techniques is then evaluated for performance /throughput improvement. Swiss-Prot database was aligned using various alignment tools. NVIDIA TESLA K40 GPU is being utilized for generating the results for this research. This implementation achieves the performance of 44.3 Giga cell updates per second (GCUPS), which is 22.9 times better than its implementation on GTX 275. Performance is improved as the workload of sequences of equal length is equally distributed among all the threads on Multiprocessors of GPU.

Keywords: Bioinformatics Sequence Alignment, High Performance/Throughput Computing, Algorithmic Implementations on GPUs

1. Introduction

Sequencing is one of the most commonly used technique in bioinformatics, used for arranging the DNA, RNA and protein sequences in order to find the similarities. Sequence alignment techniques are used for finding the best matching sequences.

Two types of alignments based on completeness;

Global alignment is a type of alignment which starts at the beginning of the sequence and add gaps until the end of a sequence is reached.

Local Alignment is a type of alignment which locate the region of highest similarity between two sequences and build the alignment outward from there.

Two types of alignment based on numbers;

Pairwise Alignment is an alignment of 2 sequences resulted from inserting gaps"- "to such extent that subsequent sequence has the same length and where each pair of residues represents a homologous position

*Corresponding author: Yahya_Bungash@yahoo.com

Multiple Sequence Alignment is a more general form of pairwise alignment where alignment is done among more than 2 sequences.

A wide range of algorithms for alignment has been developed in recent years [1] .Sequence alignment algorithms can be classified into two categories. The first category is Dynamic Programming (DP) [2] which recovers all the optimal alignments; however, such algorithms are slow and computationally very expensive. The second category is based on heuristic methods [3]–[5] which is much faster as compared to Dynamic Programming based algorithms; however, it does not guarantee to recover all the optimal alignments. In contrast to dynamic programming, which is based on exhaustive search, the second category of algorithms is based on probabilistic methods.

In this paper, an accurate method of accelerating such algorithms using GPUs has been investigated. A Swiss-prot database has been processed using GPU implemented Smith-Waterman Sequence Alignment Algorithm[6]. The first step in the process generates the alignment scores but not the actual alignment. Various available alignment tools like ssearch2 are then utilized to align the output file generated during the first step.

Performance is increased due to the sorting of the database into equal length sequence, due to which workload is distributed among all threads. This implementation achieves the performance of 44.3 Giga cell updates per second (GCUPS), which is 22.9 times better than its implementation on GTX 275. The remainder of the paper is organized as follows: Section II presents the GPU-accelerated S-W implementation for protein sequence alignment. Section III discusses the Methodology; Section IV discuss GPU-Based Approaches for pairwise/ multiple sequence alignment; Section V discuss the results and comparison of performance with other implemented methods and compares the performance of our implementation with the previous solution. Section VI concludes the paper.

2. GPU Accelerated S-W Implementation

2.1 Open-GL Approach

Before the dawning of CUDA, the Smith-Waterman algorithm was implemented on GPU using OpenGL library. Nvidia GeForce 7800 GTX card was used, the performance was increased to 0.67 GCUPS, and 10-fold speed up was achieved as compared to the CPU implementation.

2.2 The CUDA Approach

NVIDIA GPU's uses CUDA [7] which is hardware and software architecture, to execute programs coded in various programming language like, OpenCL [8], Direct Compute [9] Fortran, C,C++ and other languages. As shown in fig. 1(a), kernel that run on the GPU is called by a CUDA program [10]. In programming model, the basic unit that executes the kernel is known as thread. A kernel executes parallelly across the threads. Threads are organized by programmer in form of thread block, where each thread block consist of concurrently executing threads and for communication, they use shared memory.

GPU executes one or more than one kernel grids. Multiprocessors in GPU executes one or more thread blocks as shown in fig. 1(a). GPU can schedule multiple thread blocks to be run on multiprocessor parallelly or sequentially.

Various memory spaces exist in CUDA parallel programming model[7]. Fig. 1(b) shows complete set of CUDA memory spaces.[10]

- **Global Memory**

  GPU's RAM is used as global memory. Accessing RAM has a high latency which can be avoided using Coalescing.

- **Texture Cache**

  There is a second type of memory, which are "windows' into global memory. Such memory is called a texture cache.

- **Constant Cache**

  A constant cache is the read-only portion of the actual global memory.

- **Shared and Local Memory**

  For communication within threads of a thread block, a fast memory known as shared memory is used.

For features like function calls and also for register spills, a portion of global memory for each thread is used. This type of memory is called Local memory is basically

a b

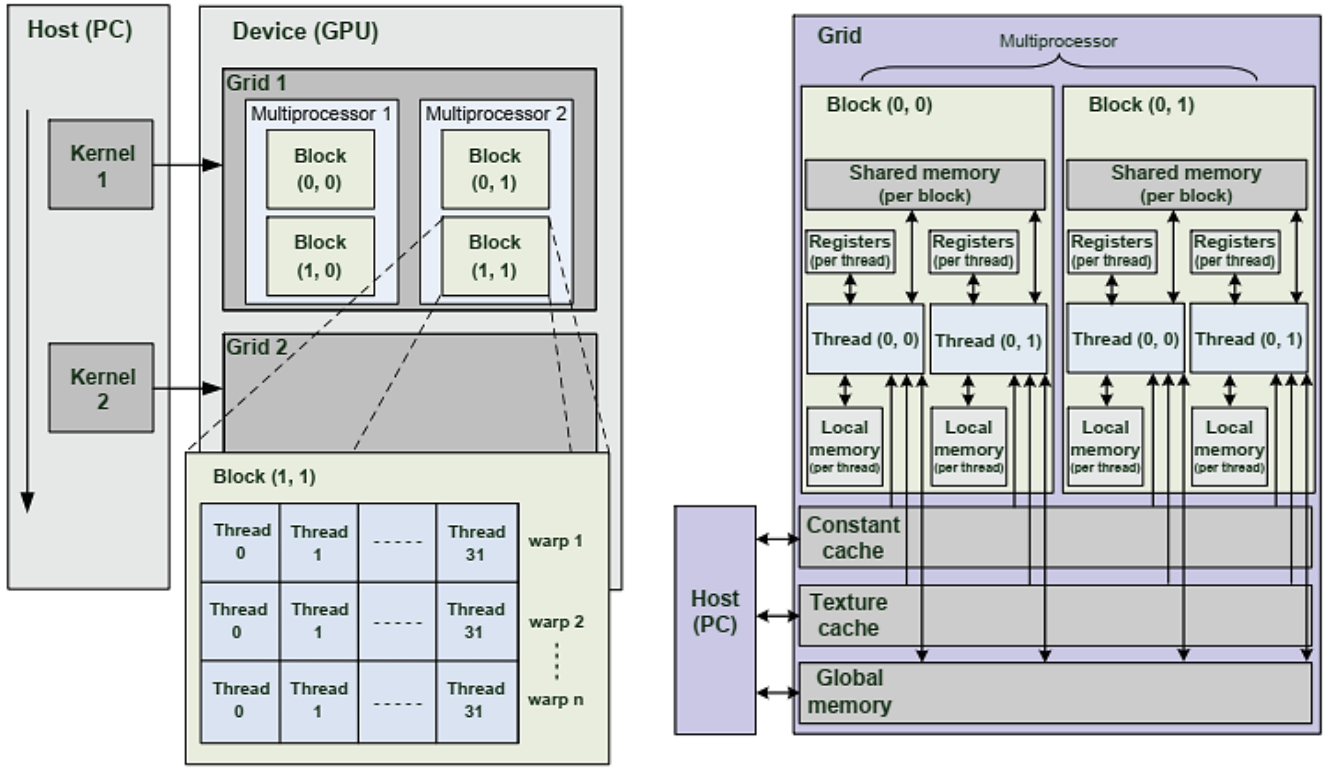

Fig 1 - (a) CUDA hierarchy of Grids, threads and blocks (b) CUDA Memory Hierarchy

Coalescing:

Latency in accessing global memory can be evaded using coalescing method as shown in Fig. 2[10] . According to coalescing, a 4-byte value in global memory is accessed by a thread of a half warp of sixteen threads. Fig. 2(a) is a non-coalesced version. It can be clearly seen that all values are stored randomly in an unordered different location. As a result, each memory access executed by each thread will be 32-byte. 64-byte and 128-byte memory access are the other possibilities.

As the value is only 4-byte and memory access is 32-byte, so 28 bytes of bandwidth per access is wasted. Total bandwidth wastage for all the memory access by 16 threads will be 28 × 16 = 448. As memory access process is a sequential process so this wastage of bandwidth will make the latency high. Fig. 2(b) shows the accessed values are stored at neighboring addresses. In this case, coalescing takes place.

A single 64-byte load is issued. As a result, no bandwidth is wasted as only single access is required.The SW-CUDA is the first acknowledged Implementation of Smith-Waterman algorithm using CUDA[11]. The method is dissimilar from the systolic array approach: a complete alignment is performed by each of the GPU's processing elements rather than them being utilized to stream through a solitary matrix.

This method eliminates the memory read and writes operation as there is no communication among these GPU processing elements. Global GPU memory store the database, for threads to have the same execution time they are sorted by length. A query profile which is an expanded substitution matrix that has query sequence elements as its column is

generated to speed up access to substitution matrix; however, rows still have the protein alphabets. CUDA, the most developed GPU programming toolkit is utilized for GPU programming (device)in combination with C++ for PC Programming(host). Protein sequence from the Swiss-Prot database is used for alignment because the alignment of protein is more complicated than DNA alignment.

Like other GPU implementation, the method used in this paper also just returned the maximum score; it does not perform any alignment. The first step in the process generates the alignment scores but not the actual alignment. Actual

Swiss-Prot database consists of more the 560,000 sequences, but this process will return on 20 top scoring sequences. Various available alignment tools like ssearch2 are then utilized to align the output file generated during the first step.

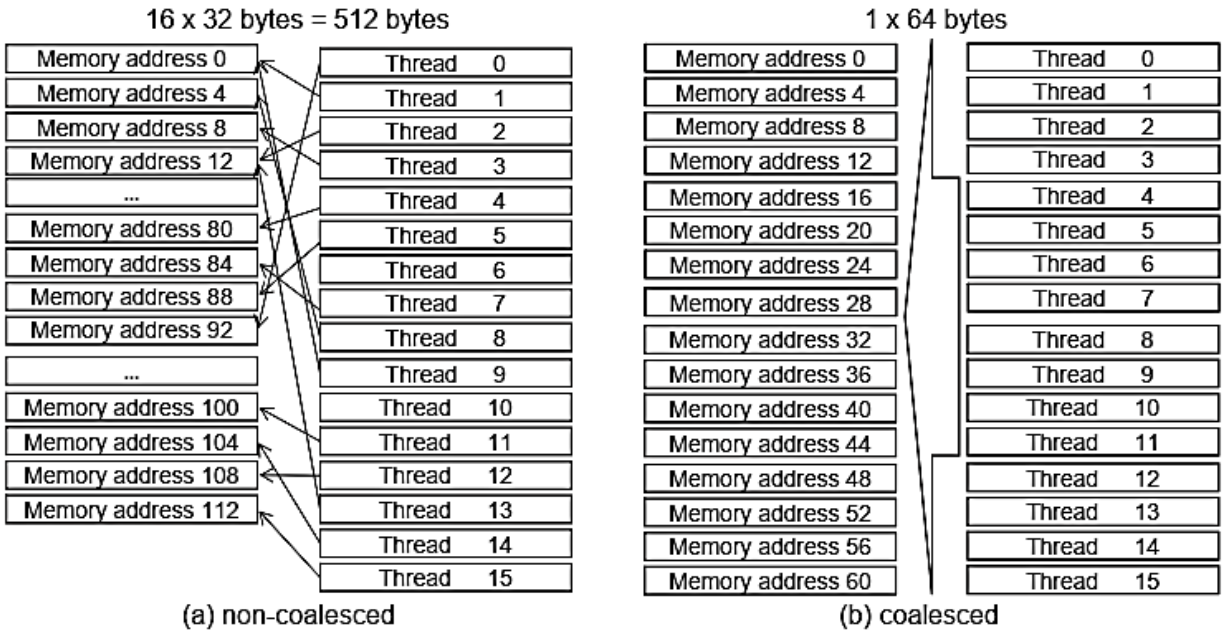

Fig.2 - This figure shows the Coalescing process

3. Methodology

Tesla K40 GPU installed in Intel Core i5 with 8gb ram is used for protein sequence alignment in this process. As discussed earlier, searching protein databases is more complicated than DNA sequences, its due to the substitution matrix. To achieve sequence parallelism, there are various techniques and methods. One method is the systolic array method which transfer sequence from one processing element to another for searching a particular sequence. The second method uses processing element to perform sequence alignment in parallel, i.e. each element aligned a sequence independently and in parallel. The method used in this paper is the second one where the processing element does not need any type of communication between them and perform complete alignment saving communication time. Tesla K40 has 2880 cores, and Swiss-Prot has more than 560,000 sequences, so it is the best way to utilize the resources and keep the cores occupied.

3.1 Database Organization

FASTA format is the default format of the Swiss-Prot database. Instead of processing the sequence in FASTA format it is first converted to custom GPU format to better match the device capabilities. The conversion process is performed only once, after which it is stored locally in GPU format. Conversion process consists of the following steps.

- **Sequence Sorting and description**:

Threads execute an instance of a program and have access to register and local memory. CUDA program executes in parallel across the threads in groups of 32 called warps. For optimization half warp concept is used where thread executes in half warp. Practically, using this method, threads in each warp must wait for each other and do not continue independently. This waiting time can be reduced by arranging the sequence by length so that neighboring threads have less length difference. Description of sequences is not uploaded to GPU to save memory and reduce load time.

- **Integration**:

After Sorting, there are still some sequences with different sizes. This issue can be resolved by integrating the residual sequences with the sequences in the sequence set to make sequence groups. Length of sequence groups is approximately equal or in ideal situation matches the length of the most extended sequence.

GPU kernel identifies the end of each sequence group through sequence terminator inserted between the concatenated sequences.

- **Interlacing:**

The output of the processed sets of sequence groups is then written into a file such that each subset consists of 8 characters from every group. Complete sets of sequence group are 16 and are arranged such that 8 characters from the first sequence group of a set are written and then the same number of characters from the second group of a set and so on.

Every thread in a half warp can now load 8 bytes as a result total of 128 bytes data loading takes place[10]

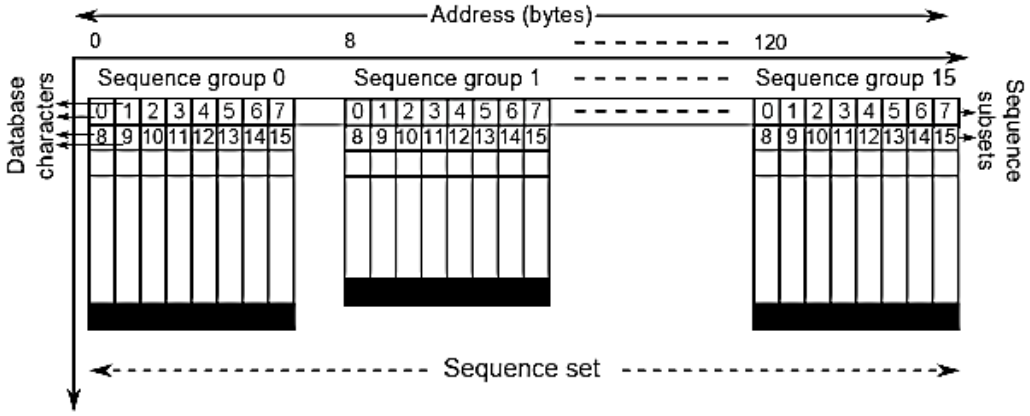

Fig. 3- This figure shows database conversion

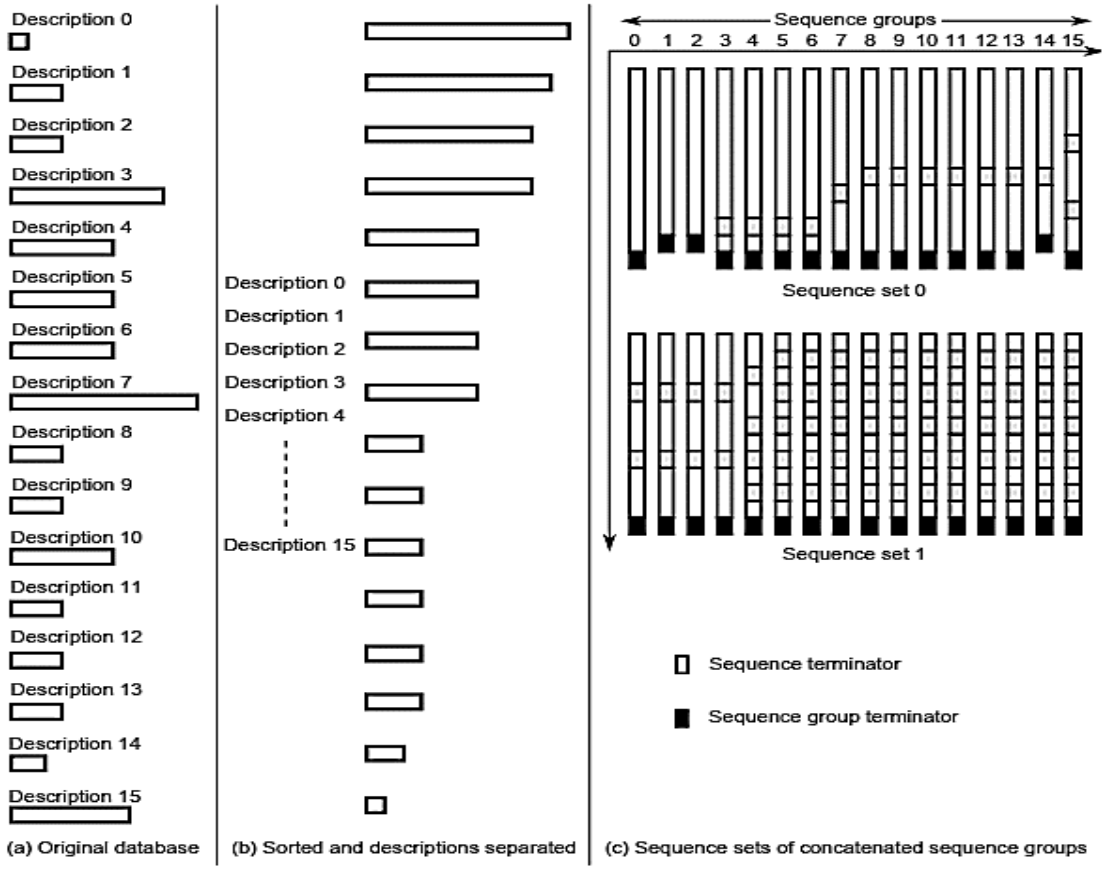

Fig. 4 - This figure shows interlacing of sequence

3.2 Temporary Data reads and writes

Number of times memory has been accessed specially for temporary data occupy too much memory bandwidth which is a serious tailback while developing GPU implementation. There is no need of saving matrix values of S-W for entire execution and can be overwritten as no traceback is performed

$$S_{i,j} = Max \begin{cases} 0 \\ S_{i-1,j-1} + W_{i,j} \\ K_{i,j} \\ Li,j \end{cases} \quad (1)$$

where

$$K_{i,j} = Max \begin{cases} S_{i-1,j-\alpha} \\ K_{i-1,j-\beta} \end{cases} \quad (2)$$

$$L\ i,j = Max \begin{cases} S_{i-1,j-\alpha} \\ L_{i,,j-1-\beta} \end{cases} \quad (3)$$

Score column saves data generated from the left side of equation 1, i.e. $S_{i,j.}$ Size of temporary data is set according to the query sequence and not the sequence database. This results in less utilization of memory as it is very unlikely that query sequence will be as long as the length of the most extended database sequence. At the start of a new database sequence, the column that store temporary data is initialized to zero. For 'K' values, another column is added. Furthermore, the upper value of 'L' is kept. Each iteration of S-W involves writing and reading temporary values of score and k for 4 accesses in total.

For each access 32 bytes read/write are issued when both are non-coalesced. Total bandwidth used is 2048 bytes (16 threads x 32 bytes x 2 values x 2 read/write) which is a significant memory bottleneck. This can be optimized to 128 bytes instead of 2048 byte as following:

- The data type used is 16 bits, which cuts the memory bandwidth required theoretically in half, which allows better coalescing.
- A pointer is used instead of direct array access for the temporary storage. Pointer is increased according to the number of threads, each thread in a half wrap reads a coalesced data of 2 bytes. Instead of accessing 32 byte what it does is that two accesses per half warp then takes place.
- Score and k value are interlaced to reduce the number of memory access to half. Score and k values can be accessed in one go in an iteration .2 bytes values in one read are accessed by a thread which results in accessing 32 bytes(16x2x2) per half warp resulting in 64 bytes coalesced access
- Two temporary values are interlaced to 128-byte accesses.

3.3 Access of substitution Matrix

Every time two symbols are aligned in an alignment process, a substitution matrix is used, making its access time very important to performance. Substitution matrix used in this implementation is BLOSUM 62, and its access is random and dependent on the database. Global memory of GPU is not suitable for frequent utilization because of its high access time. Texture memory is a cached window in global memory with lower latency and does not require coalescing and has the capability of accessing 4 values at a time.

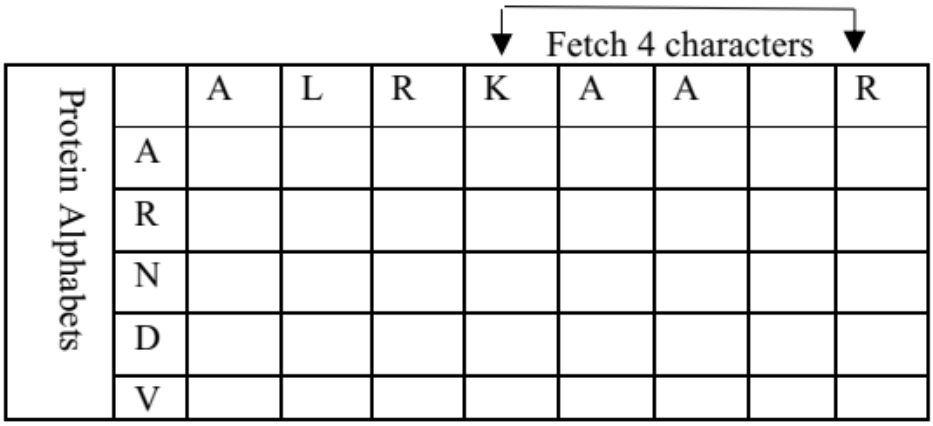

Fig. 5 - This figure show memory accesses to fetch 4 values

Mechanism discussed above can be used to fetch four values from a query profile shown in figure above. This type of substitution matrix uses query sequence instead of protein alphabets. For given character in a database, substitution matrix is not random and numerous scores can be loaded at a time during alignment process. For every query sequence a query file is generated once where each column stores 23 characters. Tesla K40 has 48KB texture memory in contrast to 8KB of GTX 275.Any query sequence of length more than 20137 characters will result in cache miss while in GTX 275,any sequence with more than 356 characters will result in cache miss[10].

4. GPU-based approaches for pairwise/multiple sequence alignment

For pairwise alignment ,a version of MUMmer Tree-based approach accelerated using CUDA is used, known as MUMmerGPU [12]. Its target is to align small query sequence with large reference sequence. CUDAlign [13] is used for pairwise alignment of large DNA sequences of Size more than a million bases. Initially an OpenGL-based multiple sequence alignment technique is deliberated in[14].

To keep the processing elements busy, parallel processing is performed for the alignment of multiple sequences. To avoid super computation sequences are sorted by length.

5. Results and Discussion

The experimental environment utilized to perform this implementation and measure its performance is as follows:

- Intel® Core™ i5-6400(2.70 GHz) with 8Gb RAM
- NVIDIA Tesla K40 graphics card with 12Gb memory and 2880 CUDA cores.
- 64-bit Microsoft Windows 10 Professional
- CUDA Toolkit version 10
- Swiss-Prot Database & BLOSUM62 Substitution Matrix

GPU accelerated S-W was implemented on advanced GPU, and the result was compared to previous implementations. Performance is measured in Giga cell updates per seconds (GCUPs) which is on average 44 GCUPS and execution time up to 24 Seconds. With the increase in Query length, the execution time increases shown in Fig. 6, but GCUPs remain almost constant. Fig. 7 shows the performance with an increase in the length of query sequences. As we can see the performance is almost constant

Our implementation is compared with implementation of same implementation on GTX 275[10], performance was increased by 22.9 times.

Fig. 8 shows a comparison of the performance of Tesla k40 and GTX 275, Tesla K40 is much better than GTX275 which clearly shows that with increase in number of cores performance is increased.

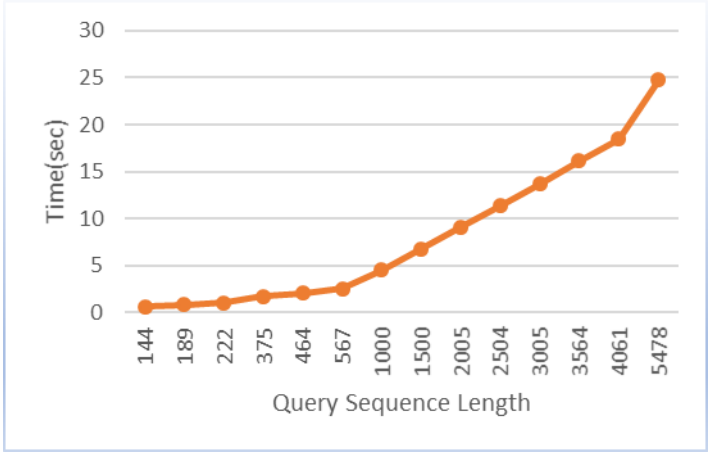

Fig. 6 - This figure shows execution time-varying with lengths

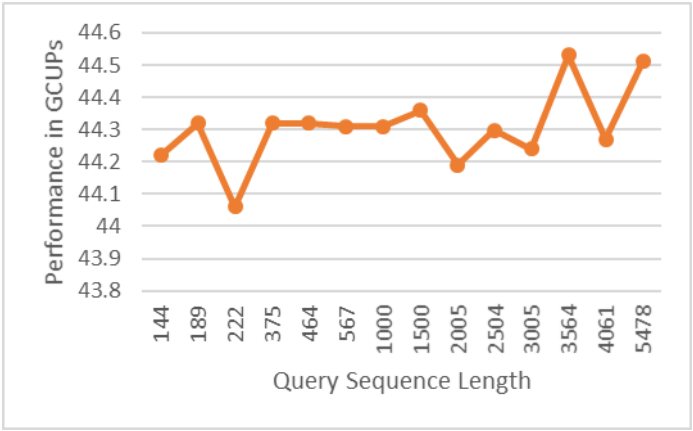

Fig. 7 - This figure shows Performance with various sequences

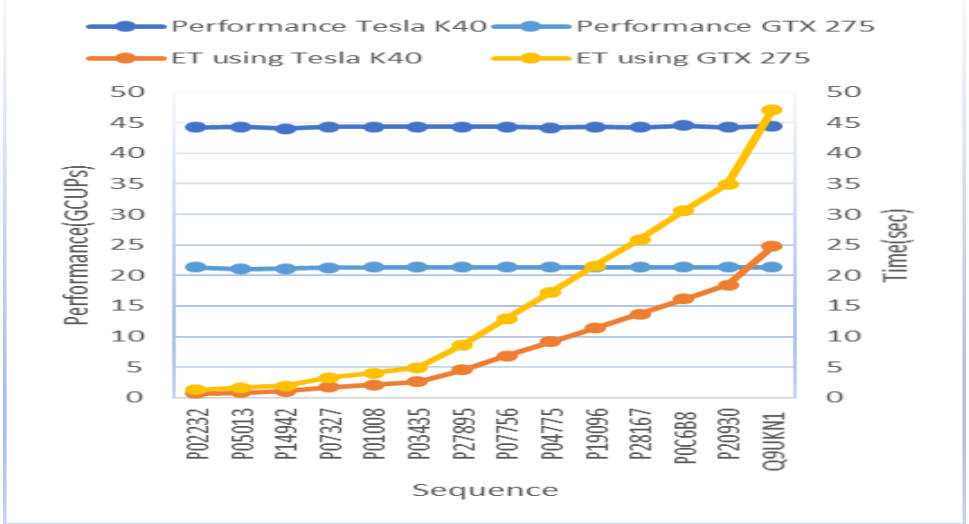

Fig. 8 - This Figure shows a comparison of K40 vs GTX 275

Table 1 - Performance Results with Swiss -Prot

Query	Length	Execution time(sec) GTX 275	Execution time(sec) K40	Performance (GCUPs) GTX275	Performance (GCUPs) K 40
P02232	144	1.24	0.65	21.35	44.22
P05013	189	1.65	0.85	21.06	44.32
P14942	222	1.93	1.01	21.15	44.06
P07327	375	3.24	1.7	21.28	44.36
P01008	464	3.99	2.1	21.38	44.32
P03435	567	4.89	2.57	21.32	44.31
P27895	1000	8.6	4.54	21.38	44.31
P07756	1500	12.91	6.81	21.36	44.36
P04775	2005	17.27	9.13	21.35	44.19
P19096	2504	21.54	11.38	21.37	44.298
P28167	3005	25.88	13.68	21.35	44.24
P0C6B8	3564	30.67	16.12	21.37	44.53
P20930	4061	34.97	18.48	21.35	44.27
Q9UKN1	5478	47.15	24.79	21.36	44.51

6. Conclusion

The performance was increased up to 22.9 times more than the previous implementation. The execution time was at max 24 sec which is very less than previous implemented DOPA. As discussed earlier, Tesla K40 has 48KB texture memory in contrast to 8KB of GTX 275. Any query sequence of length more than 20137 characters will result in cache miss while in GTX 275, any sequence with more than 356 characters will result in a cache miss. This increases the performance up to 25%.GTX 275 has 240 cores while Tesla K40 has 2880 cores which is much more suitable for achieving parallelism in the processing of big data. The database organized in eGPU has boosted the performance a lot, so for better performance, it is recommended to buy an advanced GPU instead of CPU for high computing and parallel processing tasks. The database was organized in equal length sequence, which results in equal workload distribution for all the threads hence using multicores of GPU results in less execution time and high performance. Consequences of using GPU will be high power consumption and high cost.

Acknowledgement

Special thanks to Pakistan Science Foundation for support and funding this research work.

References

[1] Hasan, L., Al-Ars, Z., Vassiliadis, S. (2007). Hardware acceleration of sequence alignment algorithms- An overview. International Conference on Design & Technology of Integrated Systems in Nanoscale Era.92–97
[2] Giegerich, R. (2000). A systematic approach to dynamic programming in bioinformatics. Bioinformatics 16, 665 677
[3] Eddy, S.R. (1998). Profile hidden Markov models. Bioinformatics 14, 755–763
[4] Lipman, D.J., Pearson, W.R. (1985). Rapid and sensitive protein similarity searches. Science 227, 1435–1441.
[5] Altschul, S.F., Gish, W., Miller, W., Myers, E.W., Lipman, D.J. (1990). Basic local alignment search tool. Journal of Molecular Biology, 215, 403–410
[6] Smith, T.F., Waterman, M.S. (1981). Identification of common molecular subsequences. Journal of Molecular Biology, 147, 195–197
[7] Hasan, L. Kentie, M. Al-Ars,Z. (2011). GPU-accelerated protein sequence alignment. 33rd Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2442-2446
[8] OpenCL - The open standard for parallel programming of heterogeneous systems, The Khronos Group, 21-Jul-2013. https://www.khronos.org/opencl/. [Accessed: 19-Nov-2019]

[9] Artificial Intelligence Computing Leadership from NVIDIA. https://www.nvidia.com/en-in/. [Accessed: 19-Nov-2019]
[10] Hasan, L., Kentie, M., Al-Ars, Z. (2011). DOPA: GPU-based protein alignment using database and memory access optimizations. BMC Research Notes 4, 261
[11] Kentie, M.A., n.d. Biological Sequence Alignment Using Graphics Processing Units 123
[12] High-throughput sequence alignment using Graphics Processing Units | BMC Bioinformatics. https://bmcbioinformatics.biomedcentral.com/articles/10.1186/1471-2105-8-474 [Accessed 21-Jan-2020]
[13] Sandes, E.F.O., de Melo, A.C.M.A. (2010). CUDAlign: using GPU to accelerate the comparison of megabase genomic sequences.ACM SIGPLAN Notices, 137–146
[14] Liu, W., Schmidt, B., Voss, G., Muller-Wittig, W. (2007). Streaming Algorithms for Biological Sequence Alignment on GPUs. IEEE Transactions on Parallel and Distributed Systems, 18, 1270–1281