PAPER • **OPEN ACCESS**

Accelerating Smith-Waterman Algorithm for Faster Sequence Alignment using Graphical Processing Unit

To cite this article: Karamjeet Kaur *et al* 2022 *J. Phys.: Conf. Ser.* **2161** 012028

Accelerating Smith-Waterman Algorithm for Faster Sequence Alignment using Graphical Processing Unit

Karamjeet Kaur[1], Sudeshna Chakraborty[2], Manoj Kumar Gupta[3]

[1]School of Engg. and Technology, Sharda University, Greater Noida, U.P. India
[2]School of Engg. and Technology, Sharda University, Greater Noida, U.P. India.
[3]Shri Mata Vaishno Devi University, Katra, J & K, India.

Email: karam_7378@yahoo.com

Abstract. In bioinformatics, sequence alignment is very important task to compare and find similarity between biological sequences. Smith Waterman algorithm is most widely used for alignment process but it has quadratic time complexity. This algorithm is using sequential approach so if the no. of biological sequences is increasing then it takes too much time to align sequences. In this paper, parallel approach of Smith Waterman algorithm is proposed and implemented according to the architecture of graphic processing unit using CUDA in which features of GPU is combined with CPU in such a way that alignment process is three times faster than sequential implementation of Smith Waterman algorithm and helps in accelerating the performance of sequence alignment using GPU. This paper describes the parallel implementation of sequence alignment using GPU and this intra-task parallelization strategy reduces the execution time. The results show significant runtime savings on GPU.

1. Introduction

The sequence alignment is a technique to arrange sequences of protein, DNA and RNA to find similarity which show evolutionary, structural and functional relationship between sequences. Sequence analysis helps in development of drugs for many diseases. Sequence alignment can be pairwise or multiple sequences alignment. In pairwise alignment, only two biological sequences are compared and it may be global or local alignment. For global alignment, Needleman Wunsch algorithm is used whereas for local alignment, Smith Waterman algorithm is used. Both algorithms are simple but are computationally intensive to align large biological sequences. These algorithms need to be modified to align large biological data sets to speed up its runtime performance. In case of multiple sequence alignment, three or more sequences are compared and it uses complex algorithms. Such algorithms are based on dynamic programming to find similarity for local and global sequence alignment. Although these two algorithms give optimal solution but cannot be applied to align large sequences because as the no. of sequences increases there is exponential increase in complexity [1]. With the introduction of GPU which helps in writing parallel programs by using multiple threads simultaneously, there is a significant improvement in performance. So, the sequential version of basic algorithms can be replaced by parallel version.

Two types of parallelization are now possible. In first, GPU's multi-core capabilities are used while in second, parallel capabilities of GPU and CPU are combined together and second approach provides better results than first. Graphic processors are programmable through CUDA and OpenCL language. When these languages are used for scientific algorithms with GPU then it gives better performance than multi-core CPU [2]. GPU hardware is used for specific computing and sequential algorithms need to modified to run on GPU. For computationally intensive problems, the performance of GPU depends on hardware capabilities like number of cores and memory.

The Smith-Waterman algorithm provides maximum sensitivity at slow speed. The biological databases are increasing at exponential rate but improvement of microprocessor is done at small rate. Special purpose hardware like super computers, field programmable gate array can be used as an alternative but are very expensive. After the release of Nvidia GPU as an efficient accelerator, Nvidia released CUDA programming language which allows to access hardware primitives of GPU directly [3] and performance is better than other commodity hardware at low cost when compared to CPU based implementation and Single instruction multiple data stream-based solution.

Parallelization of Smith-Waterman can be done at instruction level by using Single instruction and multiple data instruction in which same task can be performed on multiple data items in parallel and SIMD registers are used parallelly corresponding to query sequence [4]. Intel SSE2 functions are used to implement Smith-Waterman algorithm. The execution time is reduced by reducing the no. of instructions in the inner loop and data dependency of SIMD registers is reduced also that resulted in efficient implementation of dynamic programming.

Iterative pairwise alignment can be used for aligning multiple sequences but it is time consuming as the problem instance is increasing in short period of time. Instead of using Message passing interface on PC cluster whose maintenance cost is high, GPU can be used for high performance computing which has hundreds of cores in single chip. The parallel computing can be used with GPU architecture [5].

Field-programmable gate array also provide reconfigurable hardware platform for scanning sequence databases [6]. Smith–Waterman algorithm can be mapped using processing elements efficiently at low cost. Processing elements can be reconfigured dynamically in such a way that the available resources can be used in most effective manner. Database scanning uses partitioning strategy with varying sequence length on fixed-size processor array that results in supercomputer-like performance.

An open-source sequence alignment program is developed that runs on Graphics Processing Unit [7]. This program is written using CUDA to align sequences against a single reference sequence. The suffix tree is constructed for reference sequence on the CPU and then both query sequences and this reference sequence transferred to the GPU and aligned to the tree on the GPU.

The related work is presented in section 2, detailed methodology in section 3, result with discussion in section 4 followed by conclusion with future scope in section 5.

2. Related Work

Many algorithms and tools are designed to align sequences which provides high accuracy and faster execution time. Cedric Notredame, Desmond G. Higgins and Jaap Heringa [8] implemented T-Coffee which uses progressive alignment as a combination of global and local pairwise alignment to improve accuracy at modest speed. The pairwise alignment programs like ClustalW, Lalign are extended to improve the performance of sequence alignment. For extending these programs, position-specific scoring scheme is used instead of substitution matrix for aligning sequences.

Robert C. Edgar [9] implemented MUSCLE program for alignment of protein sequences. This uses k-mer counting for distance estimation, log expectation score for progressive alignment and tree dependent restricted partitioning for refinement. There are 3 stages. In first stage, corresponding to input sequence, first distance matrix is calculated using k-mer distance to increase the speed of alignment rather than accuracy then this matrix is clustered and first binary tree is constructed and branching order is followed in prefix order for progressive alignment. In stage 2, suboptimal tree generated by stage 1 is improved by re-estimation using Kimura distance and second distance matrix is calculated, clustered and then second binary tree is constructed similar to stage 1. If there is any change in branching order then edge is chosen from second tree, it is deleted from second tree which divide it into two subtrees. These two subtrees are realigned and new optimized alignment is produced with more accuracy.

T. Oliver, B. Schmidt, D. Nathan, R. Clemens and D. Maskell [10] modified progressive alignment based ClustalW tool. It has been identified that the first stage of calculating the distance matrix for pairwise alignment is very time consuming and needs acceleration. A flexible platform named field programmable gate array is used for parallelization strategy. The Smith-Waterman algorithm is mapped onto a linear array of processing elements. While calculating DP matrix, one recurrence relation is used which count the exact matches. Each character of primary sequence is assigned to one processing element. The secondary sequence is moved through linear chain of processing elements. Each processing element compute one value for the DP matrix. There is one limitation of reconfigurable hardware chip that is no. of processing units are fixed but sequences of amino acid are longer. So, to overcome this limitation, computation is partitioned by using a loop that connects the right processing element to left processing

unit. This parallelization strategy achieves speed up at less price.

Yongchao Liu, Bertil Schmidt, Douglas L. Maskell [11] presented MSA-CUDA program in which all three stages of ClustalW are parallelized. Stage one is parallelized by using inter-task and intra-task procedure. In inter-task parallelization, task is assigned to one thread and tasks of dimBlock are performed in parallel. In intra-task parallelization, task is assigned to a complete thread block and dimBlock threads in the thread block perform the task parallelly. For storing intermediate alignment, it needs additional memory during the distance matrix computation.

Yongchao Liu, Douglas L Maskell and Bertil Schmidt [12] optimized the search operation of database of SW algorithm using GPU. An open source CUDASW++ 1.0 is developed using low-cost GPU. Two parallel approaches named Inter-task and Intra-task are used to search sequence databases using CUDA. It is found that inter-task parallelization has better performance than intra-task but it takes more device memory than intra-task parallelization. In order to get optimal results, these two approaches are combined together as two stages. The first and second stage corresponds to inter-task and intra-task parallelization respectively. These two stages are separated by setting the threshold to 3072 which indicate the length of subject sequence. The first stage align sequence if its length is less than or equal to 3072 otherwise it is aligned in second stage.

Yongchao Liu, Bertil Schmidt, Weiguo Liu, Douglas L. Maskell [13] developed CUDASW++ 2.0 which optimize the performance of CUDASW++ 1.0 [12] by using single instruction multiple threads and partitioned vectorized algorithm. In this, query sequence is divided into multiple small partitions according to specified partition length. Each partition is consecutive and non-overlapping. The alignment scores are stored in shared memory or registers. Each partition is considered as new query sequence but while calculating alignment matrix of new partition, last row of earlier calculated matrix is considered as first row, the entries of last row in previous matrix is same as the first row of recently calculated matrix and are kept unchanged.

Waris S,Yalcin F,Jackson KJL, Nap JP [14] utilized the parallel capability of graphical processing unit in Smith Waterman algorithm. It has been proved that computational capabilities of NVIDIA based GPU can be accessed and utilized for aligning sequences. This designed algorithm produces alignment profiles with no. of gaps, score in terms of matches and mismatches. It can handle multiple hits for sequence alignments. Many cases are tested to check the versatility and usability of new parallel implementation of SW algorithm.

Warris S, Timal, Kempennar et. al [15] further extended the work of [14] because previous algorithm is platform specific. To make it platform independent OpenCL and Python language is used to parallelize and generalize SW algorithm. This algorithm can run on several hardware platforms including GPU and provides suitable details of alignment with more accuracy. It also supports Python libraries, input, output, affine gap penalty as well as logging. User friendly and developer environment is created by integrating Python with parallel computing language.

3. Methodology

This paper present how multithreaded based parallel design of Smith-Waterman algorithm mapped on NVIDIA GPU with the help of CUDA.

3.1 Smith Waterman Algorithm

The Smith-Waterman is very popular algorithm in bioinformatics to determine similar regions between two alignment sequences. These sequences which needs to align may be nucleotide or protein sequences. The alignment of two sequences starts by filling S_q * S_r matrix A. This matrix is called similarity matrix. The entries in this matrix are calculated dynamically. Here S_q and S_r represent the total no. of elements in query and reference sequence. Each value in the matrix shows similarity between two subsequences. The value in matrix A [p, q] represent up to p^{th} and q^{th} position element in query and reference sequence where 1 <= p <= S_q and 1 <= q <= S_r. The formula for calculating this similarity $A_{p,q}$ is as follows:

$$A_{p,q} = \text{Maximum}\left[A_{p-1,q-1} + \text{cost}\left(a_p, b_q\right), B_{p,q}, C_{p,q}, 0\right] \quad (1)$$

Where cost (a, b) represents cost of substitution. Values of $B_{p,q}$ and $C_{p,q}$ is calculated by

$$B_{p,q} = \text{Maximum}\,[A_{p,q-1} - x, B_{p,q-1} - y] \quad (2)$$

$$C_{p,q} = \text{Maximum}\,[A_{p-1,q} - x, B_{p-1,q} - y] \quad (3)$$

Here x and y are gap penalties for alignment of sequences. When p < 1 and q < 1 at initial stage then value for $A_{p,q}$, $B_{p,q}$ and $C_{p,q}$ is zero. The values of x and y are equal to 1 for linear gap penalty. The substitution cost (a, b) = -1 if value of a and b is unequal else cost (a, b) = 2.

We obtain matrix A in which each matrix cell is represented by $A_{p,q}$ according to p^{th} and q^{th} position. Figure (1) illustrates how matrix A can be calculated for two S_A and S_B sequences. Once the matrix calculation is over then traceback is used to find out similarity between these two sequences. Trace back process start by searching highest value from the matrix entries.

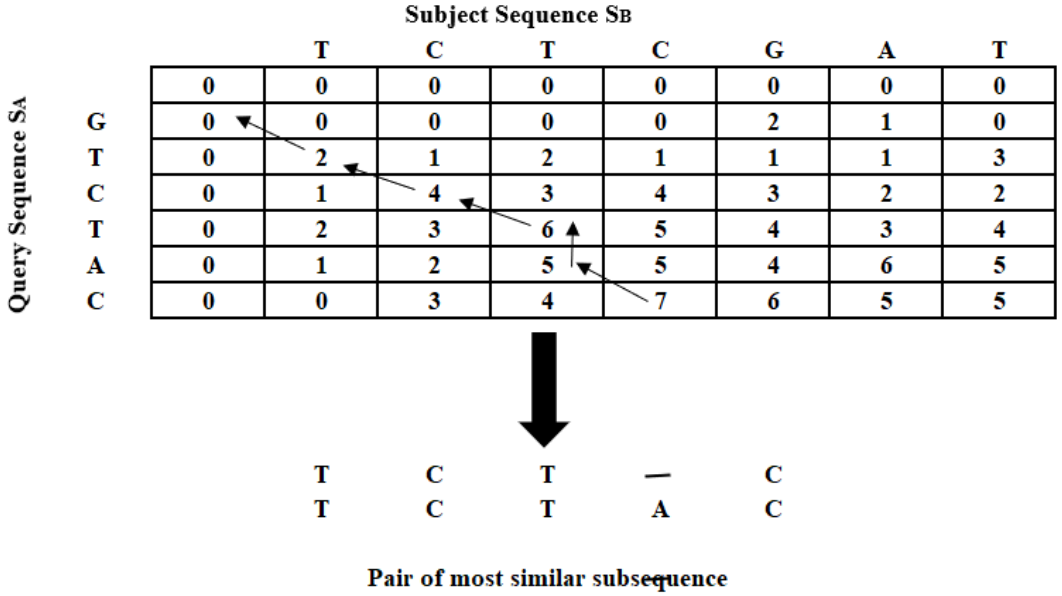

Figure 1. Similarity Matrix Calculation

The figure (1) shows the obtained matrix A corresponding to two sequences that are aligned. These sequences are S_A = G T C T A C and S_B = T C T C G A T. The similarity between subsequences is represented by entry in matrix. The highest value in this similarity matrix is 7 at $A_{4,6}$. Traceback procedure starts from this position, move in upward direction and get the best alignment result. To get best alignment of two subsequences, the trace backing starts with the highest values which is at $A_{4,6}$ and go upward direction and by using this procedure reach at the upper-left corner of the similarity matrix. According to the current position, the maximum value available at three neighbors is selected as the new position.

3.2 CUDA

NVIDIA Corporation developed CUDA as a parallel computing architecture through which software developers can use the computing engines of GPU through standard programming language. GPU is a multithreaded as well as parallel computing device.

CUDA is a development environment for GPU. Each CUDA enabled GPU has compute capability. In this paper, along with NVIDIA GEFORCE GTX 1650, CUDA compute capability version 10.2 is installed on Intel core i7 processor. CUDA allow to transfer compute intensive task from CPU to GPU. CUDA has in-built libraries for optimized code for GPU. User can write own GPU code and combine these codes with

calls to these built-in libraries. For parallel execution, CUDA uses threads and GPU allows these threads to execute parallelly at the same time. CUDA program calls kernels which execute on GPU. The kernel runs set of threads parallelly. The thread unit runs instance of the kernel. The set of multiple threads are responsible for parallel processing. The set of threads is known as thread block. When thread blocks are grouped together then they represent grid. Same code is run by each thread sequentially.

All the threads in thread block are synchronized. Threads can access registers as well as local memory. The programming model of CUDA can be seen in Figure (2).

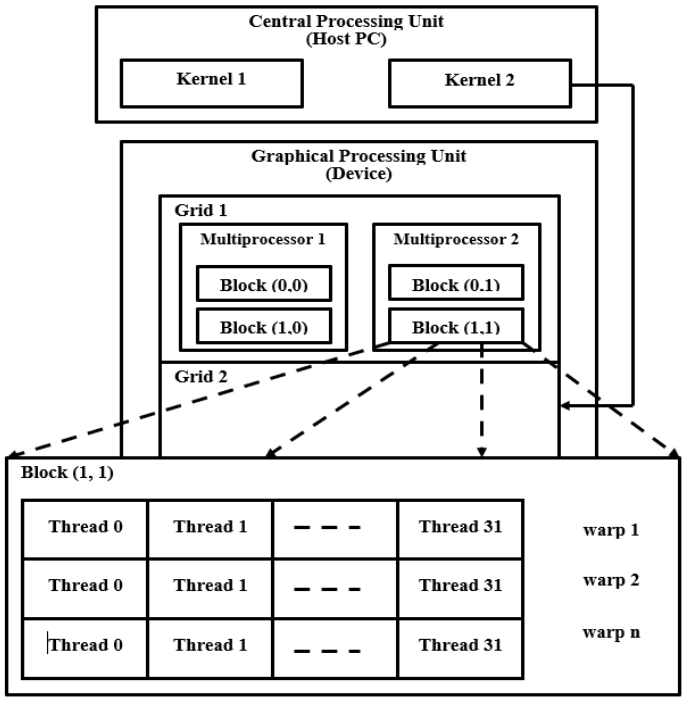

Figure 2. Programming model to represent threads, blocks and grid

The thread block runs threads concurrently and communication between the threads takes place through shared memory. A grid reads input from global memory. The grid is also responsible for synchronization between interdependent calls to kernels. The CUDA manage threads through processor available in the GPU.GPU has multiprocessor that execute thread blocks using thread switching. The processing elements in multiprocessor are called cores. These cores execute group of threads called warp.

The figure (3) shows different types of memory available in GPU that can be used by CUDA programming model. GPU's RAM is called global memory. The different types of memory are:

1. Register: This type of memory is private to each thread and it is on-chip. For one thread block it has 16 KB memory and 128-bit registers. It is fastest memory.

2. Shared Memory: This type of memory is on-chip and shared by threads in thread block. It is not very large but is fast as registers. It is 16 KB per thread block. It is used for inter-thread communication.

3. Global Memory: Its size is large typically 1GB.It is not cached so its latency is high. This type of memory is shared by threads but it is within grid.

4. Texture Memory: Its size depends on global memory. It is cached and read-only memory. This type of memory is shared by threads and it is also within grid. It has higher latency.

5. Constant Memory: Its size is small generally 64KB per grid. It is fast as registers and like texture memory, it is also read-only memory.

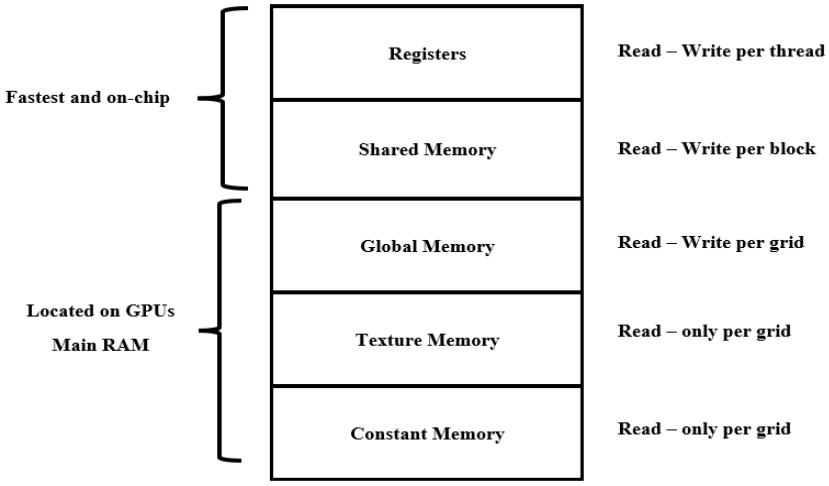

Figure 3. Memory available in GPU

The figure (4) shows the memory model. Constant memory is read only memory. It is just like cache to each multiprocessor so it is fast as register. Within thread block, inter-thread communication uses shared memory. Local memory is local to individual thread. Each multiprocessor has group of registers which are shared between its own processors [16].

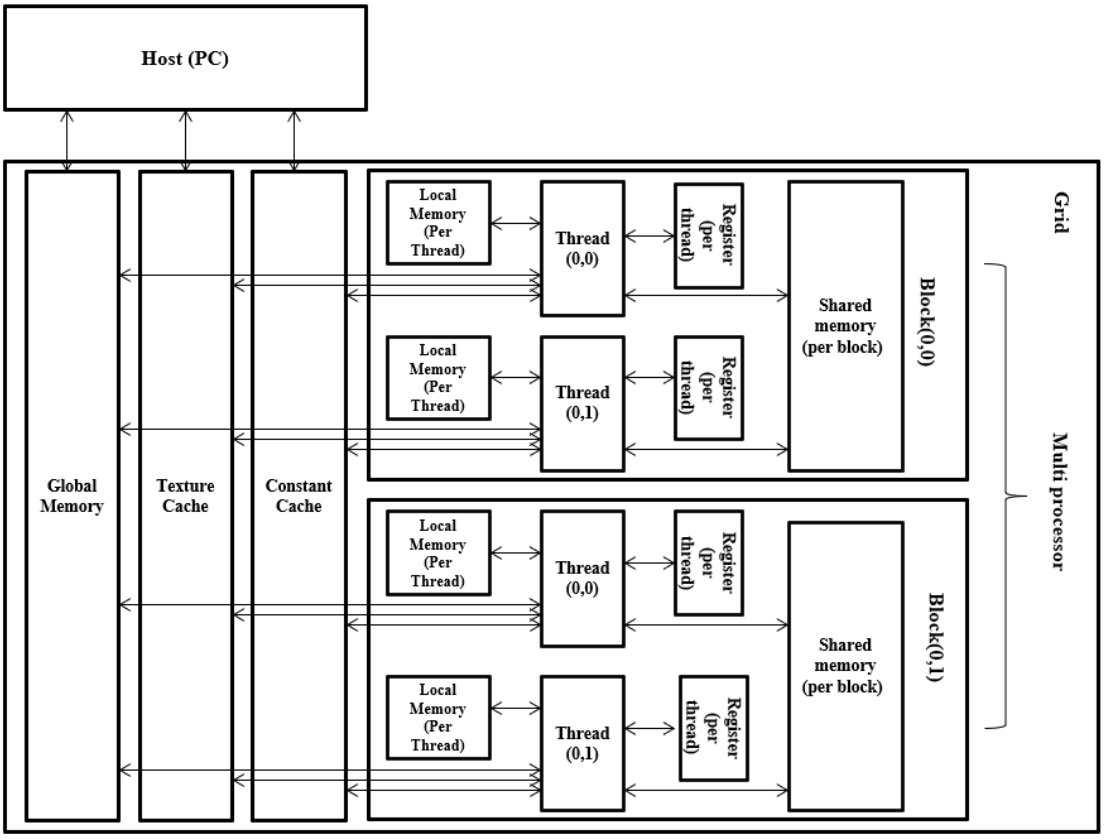

Figure 4. Memory model for CUDA memory hierarchy

3.3 Mapping of Smith Waterman on GPU

SW algorithm align query sequences with subject sequences and find subsequences pair which have high similarity. The cost of trace back is not more than the cost of matrix computation. In Smith Waterman algorithm. It has been found that traceback procedure can be done on the CPU because in Smith Waterman algorithm there is no parallelism in this computation of similarity matrix.

If there are M no. of subject sequences then there are no data dependencies in computation of M matrices between any cells on the same antidiagonal. It performs multiple sequence alignment between query sequence and M subject sequences simultaneously. M matrices are packed into a single matrix which is three dimensional and exploit the parallelism depth wise. So, GPU can compute M antidiagonals for this packed data.

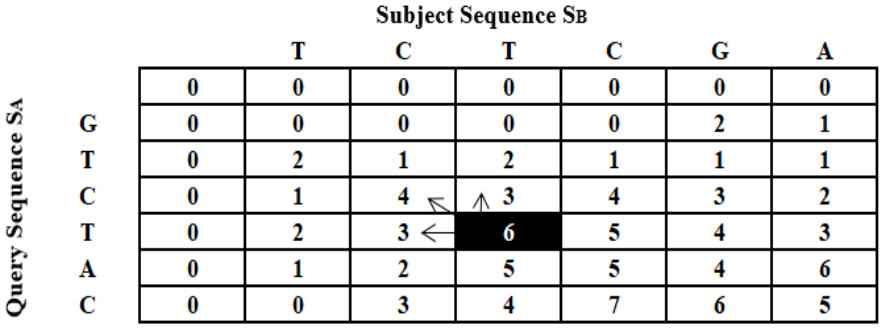

Figure 5. Matrix calculation in Sequential Smith Waterman algorithm

In figure (5) sequential approach-based Smith Waterman algorithm is demonstrated in which score is calculated from top-left corner of the matrix and moves at the bottom right. The dark entry is calculated by taking three values labelled by arrow. At a time only one value is calculated. The sequential SW algorithm is slow and is limited to specified tasks only. This dynamic programming-based algorithm is computationally slow of order (n^2).

It has been observed that all the elements which are in same antidiagonal can be calculated parallelly and independently. Data parallel sections are identified. Then identified section is considered as kernel. The range of computation is specified to invoke a kernel. This approach is motivated due to increasing power of the GPU and can be utilized as computational platform which can enhance the performance of those alignment algorithms which are based on dynamic programming. GPU's architecture opens up new possibility for sequence alignment with enhanced programmability of GPU. In figure (6) Smith Waterman parallel approach is demonstrated in which score for multiple cells is calculated in parallel. All the black cells are calculated by taking three values simultaneously in parallel.

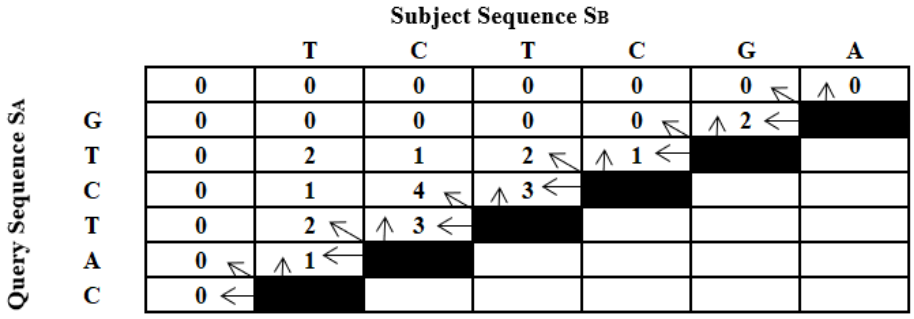

Figure 6. Matrix calculation in parallel Smith Waterman Algorithm

The procedure for alignment is like this. There are n sequences and length of each sequence is x. These are available on horizontal axis and horizontal axis represents sequencing platform. If the length of sequence

is shorter than x then special characters are added to the sequence to make its length equal to x. All the sequences are considered as a single string of length n * x. The target sequences are placed on vertical axis. In the same way, there are m target sequences, each sequence is of length y.

If the length of sequences is shorter than y then special characters are also added to the sequence to make its length equal to y. All sequences are also considered as single string of length m * y. These two strings n and m then transferred to global memory available in GPU. Alignment starts from top left but values in matrix are calculated in parallel over the diagonal of the matrix. It is necessary to find out maximum value which helps in trace backing procedure which is itself in parallel way. Trace backing starts in reverse order from bottom right to top left.

4. Result and discussion

This section presents the experimental setup used to find results, parallelization technique, performance evaluation, comparison of CPU and GPU based implementation and speed up achieved.

4.1 Experimental setup

The following hardware and software setup is used to measure the performance of sequence alignment in terms of execution time:
Intel Core i7 9750H 2.60GHz
16 GB RAM
NVIDIA GeForce GTX series 1650 graphic card
Graphic card has 8151 MB of memory with clock speed of 1560 MHz
Video driver version 417.98
CUDA Cores 1024 with memory data rate 8 Gbps and memory interface of 128 bit
Linux 64-bit operating system with Ubuntu distribution version 16.04 LTS
1 TB SDD
CUDA toolkit version 10.2
Python version 3.6.9
Substitution Matrix BLOSUMS62

4.2 Performance comparison

The performance of this strategy is evaluated in terms of execution time under two cases. In first case CPU is used to align sequences. The table (1) and figure (7) demonstrate that the execution time taken by CPU is increases as the no. of sequences increases.

In the second case GPU is used to align the sequences and performance is evaluated on same no. of sequences. The result of table illustrates the execution time taken by GPU to align the sequences. To evaluate this implementation, Intel i7 CPU is used as a host machine. This host machine has NVIDIA GeForce GTX 1650 GPU. The host machine runs Ubuntu 16.04, CUDA toolkit 10.2.

Then execution time was found for different length of sequences. The equation is used to calculate the improvement by the GPU over CPU. The improvement is given by speed up ratio.

Speed up ratio is defined as the ratio of serial to parallel execution time. In this case it is the ratio of time taken by CPU to GPU. If serial application executes in 825 seconds and corresponding parallel application runs in 246 seconds (using 64 threads and 1024 cores in this case), the speed up of parallel application is 3.4X (825/246 = 3.4).

$$\text{Speed up} = \frac{\text{Time taken by CPU (Serial version)}}{\text{Time taken by GPU (Parallel version)}}$$

If the length of sequences is large then it will take more time to transfer the data between CPU and GPU. The performance of GPU is increased by 3 times as compared to CPU performance. The results in table (1) shows GPU performance grows faster than CPU. So, the combination of GPU hardware with suitable programming language like CUDA leads to a significant improvement in performance. The proposed

approach can be seen as accelerator which satisfying the computationally intensive problem by low-cost solution. All modern computer has access to these graphic cards. Query sequences have been selected to test the performance.

Table 1. Comparison of sequence alignment of CPU and GPU based implementation of Smith Waterman Algorithm

No. of Biological Sequences	Time taken by CPU (minutes:seconds) Platform: INTEL Framework:OpenCL	Time taken by GPU (minutes:seconds) Platform: NVIDIA Framework: CUDA
20	01:17	00:23
40	02:11	00:34
60	03:07	00:51
80	03:44	01:09
100	04:00	01:19
120	04:05	01:29
140	04:19	01:42
160	05:45	01:49
180	06:49	02:12
200	07:21	02:25
220	08:18	02:42
240	09:09	02:54
260	09:43	03:02
280	10:02	03:09
300	11:42	03:25
320	12:17	03:51
340	13:45	04:06

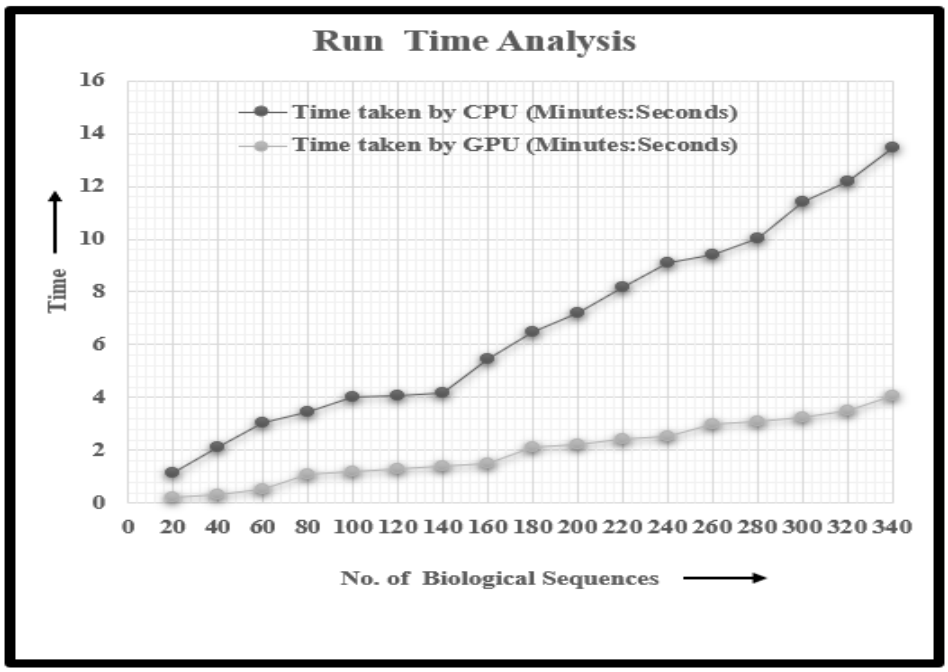

Figure 7. Comparison between execution time taken by CPU and GPU to align sequences

5. Conclusion and future scope

GPU is very attractive solution for enhancing the performance. Computer's computing capability can be increased by adding GPU to CPU. The important task of sequence alignment is combined with parallel architecture of GPU. The sequential approach-based Smith Waterman is most commonly used algorithm for sequence alignment but due to its time-consuming process of calculation, it cannot be used to align large sequences. This paper proposed parallel approach of SW algorithm on GPU with CUDA. Its 64 threads are responsible for multithreading which helps to calculate sub-matrices in parallel and after calculating alignment score, trace backing procedure to find best possible alignment is performed parallelly also. Such type of parallelization approach helps to accelerate the process of sequence alignment and shows three times speed up as compared to sequential approach when run on NVIDIA GeForce GTX 1650 GPU on Intel i7 9750H 2.60 GHz CPU with 16 GB RAM. The performance of algorithm can be evaluated on new graphic cards also and can be further enhanced by using multiple GPUs which increase more parallelism.

References

[1] Alawneh L, Shehab MA, Al-Ayyoub M, Jararweh Y, Al-Sharif ZA. A scalable multiple pairwise protein sequence alignment acceleration using hybrid CPU–GPU approach. Cluster Comput. 2020 Dec;23(4):2677–88.
[2] Khajeh-Saeed A, Poole S, Blair Perot J. Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors. Journal of Computational Physics. 2010 Jun;229(11):4247–58.
[3] Manavski SA, Valle G. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC Bioinformatics. 2008 Mar;9(S2):S10.
[4] Farrar M. Striped Smith-Waterman speeds database searches six times over other SIMD implementations. Bioinformatics. 2007 Jan 15;23(2):156–61.
[5] Hung C-L, Lin Y-S, Lin C-Y, Chung Y-C, Chung Y-F. CUDA ClustalW: An efficient parallel algorithm for progressive multiple sequence alignment on Multiple GPUs. Computational Biology and Chemistry. 2015 Oct;58:62–8.
[6] Oliver TF, Schmidt B, Maskell DL. Reconfigurable architectures for bio-sequence database scanning on FPGAs. IEEE Trans Circuits Syst II. 2005 Dec;52(12):851–5.
[7] Schatz MC, Trapnell C, Delcher AL, Varshney A. High-throughput sequence alignment using Graphics Processing Units. BMC Bioinformatics. 2007 Dec;8(1):474.
[8] Notredame C, Higgins DG, Heringa J. T-coffee: a novel method for fast and accurate multiple sequence alignment 1 1Edited by J. Thornton. Journal of Molecular Biology. 2000 Sep;302(1):205–17.
[9] Edgar RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucleic Acids Research. 2004 Mar 8;32(5):1792–7.
[10] Oliver T, Schmidt B, Nathan D, Clemens R, Maskell D. Using reconfigurable hardware to accelerate multiple sequence alignment with ClustalW,Bioinformatics 2005 Aug15;21(16):3431–2
[11] Yongchao Liu, Schmidt B, Maskell DL. MSA-CUDA: Multiple Sequence Alignment on Graphics Processing Units with CUDA. In: 2009 20th IEEE International Conference on Application specific Systems, Architectures and Processors [Internet]. Boston, MA: IEEE; 2009 [cited 2021 Jul 26]. p. 121–8. Available from: http://ieeexplore.ieee.org/document/5200019/
[12] Liu Y, Maskell DL, Schmidt B. CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. BMC Res Notes. 2009;2(1):73.
[13] Liu Y, Schmidt B, Maskell DL. CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. BMC Res Notes. 2010 Dec;3(1):93.
[14] Warris S, Yalcin F, Jackson KJL, Nap JP. Flexible, Fast and Accurate Sequence Alignment Profiling on GPGPU with PaSWAS. Zhang M, editor. PLoS ONE. 2015 Apr 1;10(4):e0122524.
[15] Warris S, Timal NRN, Kempenaar M, Poortinga AM, van de Geest H, Varbanescu AL, et al. pyPaSWAS: Python-based multi-core CPU and GPU sequence alignment. de Brevern AG, editor. PLoS ONE. 2018 Jan 2;13(1):0190279.

[16] Hasan L, Kentie M, Al-Ars Z. DOPA: GPU-based protein alignment using database and memory access optimizations. BMC Res Notes. 2011 Dec;4(1):261.