



Analyzing The Performance Of Smith Waterman Algorithm Using GPU

A Dissertation II
Submitted

By

Azhar Ali
(11304555)

To

Department of Computer & Science Engineering

In partial fulfilment of the Requirement for the
Award of the Degree of

**Master of Technology in Computer Science
Under the guidance of**

Balwant Ram
(Assistant Professor, Lovely Professional University)
(MAY 2015)

PAC APPROVAL



School of Computer Science and Engineering

DISSERTATION TOPIC APPROVAL PERFORMANCE ✓

Name of the Student: Azhar Ali

Registration No: 11304555

Batch: 2013

Roll No: A21

Session: 2014

Parent Section: K2308

Details of Supervisor:

Designation: Assistant Professor

Name: Balwant Ram

Qualification: M.Tech

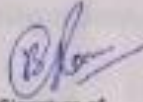
U.ID: 15584

Research Experience: 3.5 Years

SPECIALIZATION AREA: System Architecture and Design (pick from list of provided specialization areas by DAA)

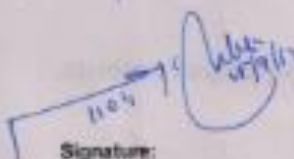
PROPOSED TOPICS

1. Parallel Computing using GPU
2. Distributed Computing
3. E-Learning Protocols


Signature of Supervisor

PAC Remarks:

Topic 1 is approved

11/04/14


APPROVAL OF PAC CHAIRPERSON:

Signature:

Date:

*Supervisor should finally encircle one topic out of three proposed topics and put up for approval before Project Approval Committee (PAC)

*Original copy of this format after PAC approval will be retained by the student and must be attached in the Project/Dissertation final report.

*One copy to be submitted to Supervisor.

ABSTRACT

With the latest advancements in the processor architectural technologies has led to the development of General Purpose Graphical Processing Units (GPU) which can be now days used to accelerate various time consuming compute intensive scientific applications. This work presents the analysis of Smith Waterman Algorithm (SWA) based on GPU computing. Smith waterman algorithm is basically used to perform the alignment of any two or more biological sequences. In present work we have tested Smith waterman algorithm for the alignment of genome sequence with H1N1 virus. GPU parallelism has been exploited using OpenCL and CUDA technologies using NVIDIA GPU GeForce 830M. The OpenCL has been used on visual studio and CUDA has been deployed with MATLAB tool. The results shows a significant speed improvement of 2.5X with OpenCL and 1.5X with CUDA platform. Speedup has been good using OpenCL with Visual Studio in comparison to CUDA with MATLAB on the similar kind of GPU.

CERTIFICATE

This is to certify that **Azhar Ali** has completed M.Tech dissertation proposal titled **Analyzing The Performance Of Smith Waterman Algorithm Using GPU** under my guidance and supervision. To the best of my knowledge, the present work is the result of her original investigation and study. No part of the dissertation proposal has ever been submitted for any other degree or diploma. The dissertation proposal is fit for the submission and the partial fulfilment of the conditions for the award of M.Tech Computer Science &Engineering.

Date:

Signature of Advisor

Name: Balwant Ram

UID:

ACKNOWLEDGEMENT

I would like to present my deepest gratitude to **Mr Balwant Ram** for his guidance, advice, understanding and supervision throughout the development of this dissertation study. I would like to thank to the **Project Approval Committee members** for their valuable comments and discussions. I would also like to thank to **Lovely Professional University** for the support on academic studies and letting me involve in this study.

(Azhar Ali)
Reg no .1130455

DECLARATION

I hereby declare that the dissertation proposal entitled **Analyzing The Performance Of Smith Waterman Algorithm Using GPU** for the M.Tech degree is entirely my original work and all the ideas and references have been dully acknowledged. It does not contain any work for the award of any other degree or diploma.

Date: _____

Investigator: Azhar Ali
Regn. No. 11304555

Table of Contents

Contents	Page No
INTRODUCTION	1
1 ARCHITECTURE OF GPU	3
2 GPU COMPUTING.....	5
2.1 THE GPU PROGRAMMING MODEL	5
2.2 GENERAL-PURPOSE COMPUTING ON THE GPU.....	5
3. SOFTWARE ENVIRONMENTS	6
3.2 OpenCL.....	7
4. GPU Programming in MATLAB	8
5. ALGORITHM	13
5.1 SMITH WATERMAN	13
REVIEW OF LITERATURE	16
PRESENT WORK.....	26
RESEARCH METHODOLOGY	27
RESULTS AND DISCUSSION	29
6.1 Gathering Required Data	29
6.2 Analysis of Smith Waterman Algorithm	31
6.2.1 CUDA USING “MATLAB”	31
6.2.2 OPENCL WITH VISUAL STUDIO	34
CONCLUSION AND FUTURE SCOPE	42
REFERENCES	43
APPENDIX.....	45
PUBLICATIONS.....	46

-Table of figure

Figure 1 GPU processing	3
Figure 2 working cores GPU and CPU	4
Figure 3 CUDA working	7
Figure 4 heterogeneous system	8
Figure 5 : Comparison of the number of cores on a CPU system and a GPU	9
Figure 6 An example of database sequences conversion. a Original database sequences, b sorted database	17
Figure 7: Flow diagram.....	27
Figure 8: Property of GPU	29
Figure 9: No of Cores in GPU	30
Figure 10: H1N1 Sample	30
Figure 11: Human Being DNA sample.....	31
Figure 12: Results in GIGAFLOPS	32
Figure 13: Alignment and Score	32
Figure 14: SW algorithm matrix	33
Figure 15: Benhmark Results	33
Figure 16: Three Selections	34
Figure 17: Selection 1 on CPU	35
Figure 18: Selection 1 on GPU	35
Figure 19: Performance on CPU and GPU	36
Figure 20: Selection 2 results on CPU	37
Figure 21: Selection 2 results on GPU.....	37
Figure 22: Performance selection 2 on CPU and GPU.....	38
Figure 23: Selection 3 on CPU	39
Figure 24: Results of Selection 3 on GPU	39
Figure 25: Performance based on Selection 3	40
Figure 26: OpenCL avg time performance on devices	41
Figure 27: CUDA giga flops average performance on devices	41

INTRODUCTION

In recent years the traditional method of improving the performance of CPUs, namely by increasing the clock frequency has exhausted its potential and performance growth is achieved by increasing the number of computing cores and the size of the on-chip cache. Current top of the line traditional CPUs are built with four cores and eight core designs are already in development. In the near future one can expect designs with more than 30 cores. The future computers will be built around processors with hundreds, or even thousands of cores. One should note that chips with comparable number of computing cores are already present on the market in large quantities. Current generation of graphic cards, such as NVIDIA GeForce or ATI Radeon contain hundreds of computing cores. The peak performance of these cards is at least one order of magnitude higher than that of traditional CPUs and they are capable of performing large scale computations for all problems where data parallel approach is feasible. Algorithms for several suitable problems have been already implemented for these platforms; in many cases showing a very good performance. The examples from diverse disciplines, such as quantum chemistry, computational fluid dynamics, astrophysics, computer science as well as search of similar sequences of biological macromolecules have been recently reported. All these examples fall into categories, which have been mentioned in the report of the Berkeley group as most challenging for the new massively parallel paradigm for software development. Therefore one may argue, that General Purpose GPU computing is the first step towards this new paradigm.

In which large computation blocks of data done with parallel. The word of GPU was promoted by NVidia in 1999, they developed the GeForce 256 as "the world's first 'GPU', a single-chip processor with integrated convert, light, triangle setup, and reading engines that are capable of processing a less of 10 million polygons per second". In a personal computer, a GPU can be present on a video card. The purpose of GPU is for the class of specific applications having characteristics as follows. From the past few years, a certain section has recognized several different applications having uniform features and represented those apps into the graphics processing unit in the proper manner. [1]

- There are huge computational requirements. There is a need of billions of pixels per second in a real-time version, and there is need of hundreds or more operations for each pixel. It is required of GPUs to render a huge measure of computing performance to require the need of tough real-time applications.
- Having parallel processing is appropriate of graphical pipeline. Some operation like vertical fragment are nicely matched to fine gained nearly parallel programming compute units, which can be applied to several other computational domain.

GPU is impotent for development it's a general-purpose computing engine. It's advantage of programing model and programming tools. The challenges of GPU developers and researcher is the proper balance amid low-level ingress to hardware to license realization and high-level programing language for flexibility. As of late GPU could be portrayed as a scholarly activity. As a result of the fundamental conduct of the apparatus and methods, the original of were checked for simple working by any stretch of the imagination. As the field developed, the methods got to be more develop and the examinations with non- GPU work more precise. Some as of late study of the recorded condenses this time of GPU figuring. We are currently entering the third phase of GPU registering, and construct some application which supportive for us. [1]

1 ARCHITECTURE OF GPU

The Graphical processing unit has all times been a processor expansive computational assets. Most weak believe is convey to light computational to the software engineer. In the course of the last few year, GPU has just constrained undertakings or capacities, with utilize some uncommon reason processor into an undeniable parallel programmable processor with extra altered capacity unique reason usefulness. [1]



Figure 1 GPU processing

AMD and NVIDIA build architecture with brought together, enormously parallel programmable units at their centre. The NVIDIA GeForce 8800 GTX (top) highlight 16 spilling multiprocessor, every contains imparted guideline and information reserves rationale, a 16 kb imparted memory, eight stream processor and two uncommon capacity

unit. A different approach is taken by GPUs. The GPU devices use different resources of the processor in various diverse stages, for an instance; say in the pipeline, it isolates in space, not in time. The piece of the processor is dealing with a solitary stage that sustains yield specifically into an assorted part that fundamentally lives up to expectations in the following stage. The course of action of GPU machine was to a great degree powerful in a GPUs settled capacity for two of the accompanying reasons.-

- Firstly- Data parallelism could be exploited by the hardware in any particular stage. Inside that particular stage, preparing of numerous components is done in the meantime. The same number of parallel stages are running in the meantime, the GPU could address numerous issues of huge PC of the representation pipeline.
- Secondly- The equipment of every stage could be changed with the assistance of unique reason equipment for the particular assignment, permitting widely more prominent competences and additionally of range proficiency over a particular reason arrangement [1]

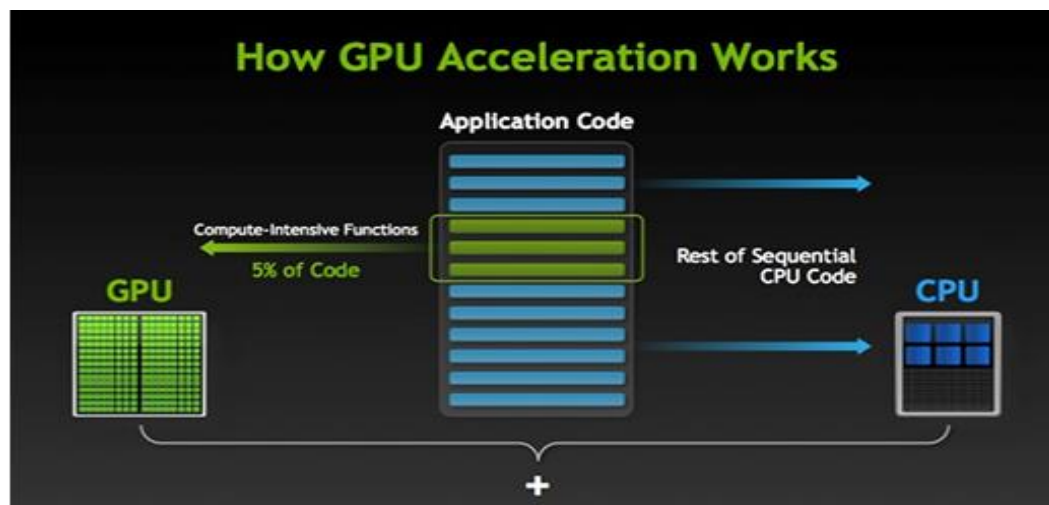


Figure 2 working cores GPU and CPU

2 GPU COMPUTING

2.1 THE GPU PROGRAMMING MODEL

The GPU programmable units follow SPMD programmable unit. With the same program the efficiency is improved by GPU while holding many elements with it. Every element being independent from one another and the elements do not communicate in the base programming model. Components read information from an imparted memory that is worldwide and to the progressed GPUs can keep in touch with the memory too. The support for SMPD model by GPU remains the question to be answered.

The classical GPU's had the advantage that it allowed unique execution path for every element that requires a substantial control over hardware. Today's GPUs support arbitrary control flow per thread but enforce a disadvantage for coherent branching. Elements are categorized into blocks and parallel processing is done on blocks. The GPU programs are written such that provision of branches exists but not free. Programmers structure their code such that they have coherent branches and make best use of hardware. [1]

2.2 GENERAL-PURPOSE COMPUTING ON THE GPU

It include the following-

2.2.1 PROGRAMMING GRAPHIC PROCESSING UNIT FOR GRAPHICS

The programmable aspects of the GPU pipeline are enlisted as

- 1) The programmer states a geometry that engulfs a region on screen. The rasterizer produces a fragment at every pixel on screen included in that geometry.
- 2) The fragment program shades every fragment.
- 3) The fragment value is computed by the fragment program by the combination of math operations and global memory that itself reads from a global "texture" memory.
- 4) The resultant picture on impending passes will be utilized as surface through the representation pipeline.

2.2.2 PROGRAMMING A GPU FOR GENERAL-PURPOSE PROGRAMS (OLD)

Making the pipeline to perform general-purpose computation, a different terminology but exactly the same steps are followed.

- 1) A geometric primitive is established by the programmer covering a computation domain of interest. At every pixel a fragment is generated by the rasterizer.
- 2) Every single section is shaded by SMPD part.
- 3) The estimation of piece is processed by the section program.
- 4) For the future passes the subsequent cradle that is in worldwide memory is utilized.

2.2.3 PROGRAMMING A GPU FOR GENERAL-PURPOSE PROGRAMS (NEW)

The organizing of GPU processing applications is done in the accompanying way

- 1) A calculation space of hobby is characterized as the structure network of threads.
- 2) Value of each string is processed by SPMD general –purpose program
- 3) Each string worth is processed by mix of math operations and both "accumulate" and "diffuse" gets to worldwide memo. [1]

3. SOFTWARE ENVIRONMENTS

3.1 CUDA

CUDA stands for compute unified system architecture, an NVIDIA created a parallel computing and programming model and employed by GPU that they produce. A direct access is to virtual instruction set is provided in CUDA GPUs. An approach called CPGPU is achieved by GPUs using CUDA. GPUs execute many concurrent threads slowly rather than a single quick execution of a thread. The CUDA in addition of being available through CUDA accelerated libraries are extensions to industry standard programming languages include C, C++, and FORTRAN.

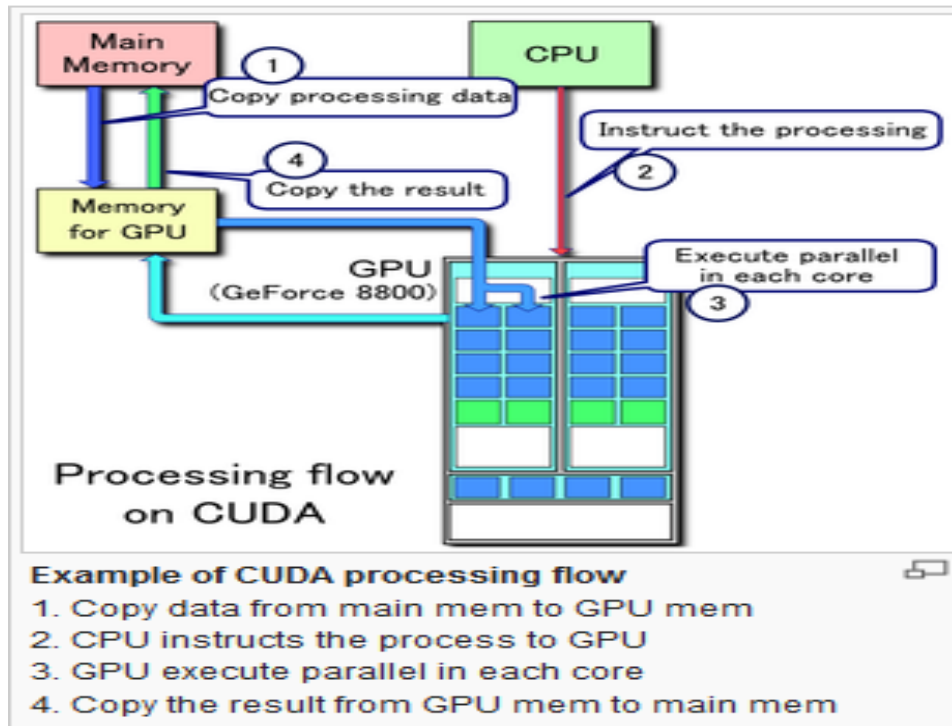


Figure 3 CUDA working

3.2 OpenCL

OpenCL is a platform that makes some type of parallel processing easy and parallel execution of code. OpenCL or open computing language is the first that was open and royalty free meant for the parallel programming for heterogeneous systems. It provides a single programming environment to write portable, efficient code for high performing computer servers, desktop systems and handheld devices using a mixture of multiple core processors CPUs, GPUs and DSPs. It defines a language in which “kernels” are written. These constitute functions capable of running on multi-variant computer devices. The computer kernels are written in an extended C language defined by the OpenCL. On-the-fly compilation of application helps the host application to take the advantage of all the compute device in the system with a unit set of portable computer kernels. [2]

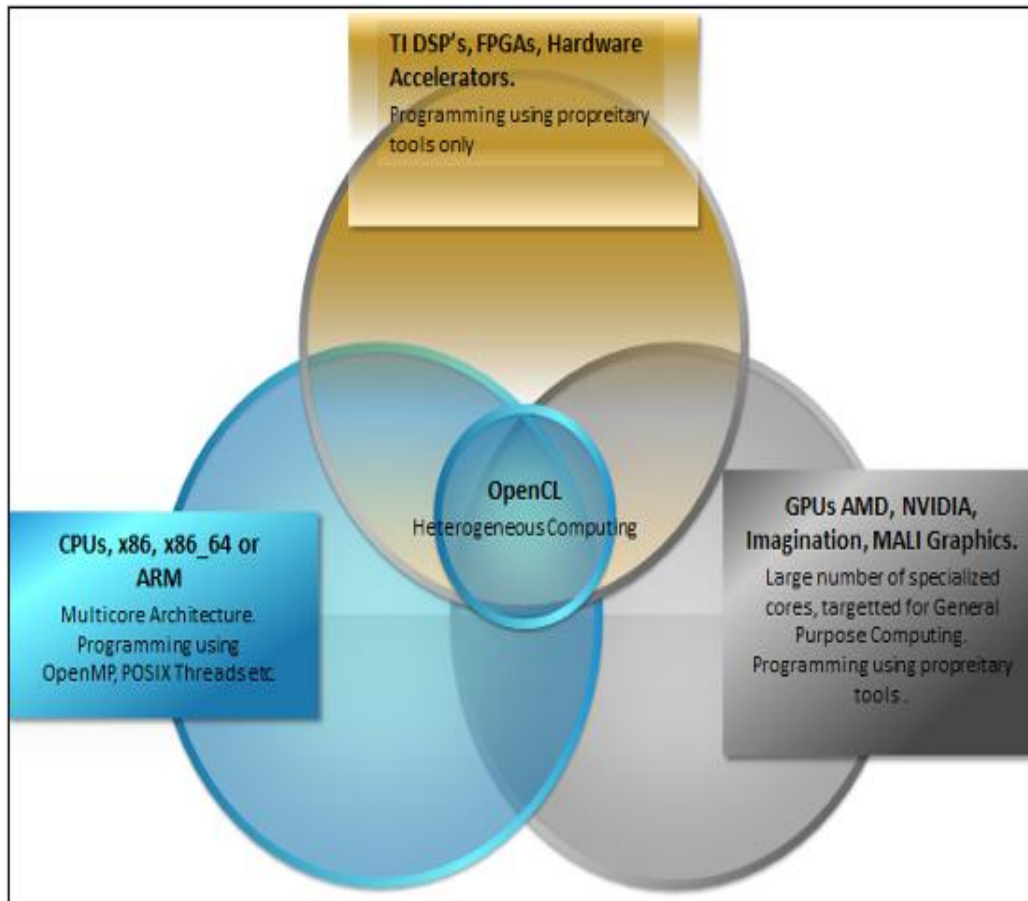


Figure 4 heterogeneous system

4. GPU Programming in MATLAB

Machines with multi cores and simultaneous-multithreading innovation has empowered researchers, engineering people, and monetary investigators to accelerate computationally escalated applications in an assortment of controls. Today, another kind of equipment guarantees much higher computational execution: the representation handling unit (Graphical Processing Unit).

Initially utilized to quicken design representation, Graphical Processing Units are progressively connected to investigative computations. Not at all like a conventional CPU, which incorporates close to a modest bunch of centers, a Graphical Processing Unit do have a hugely parallel cluster of number and drifting processors, and in addition devoted, fast memory. An ordinary GPU includes many these littler processors [3]

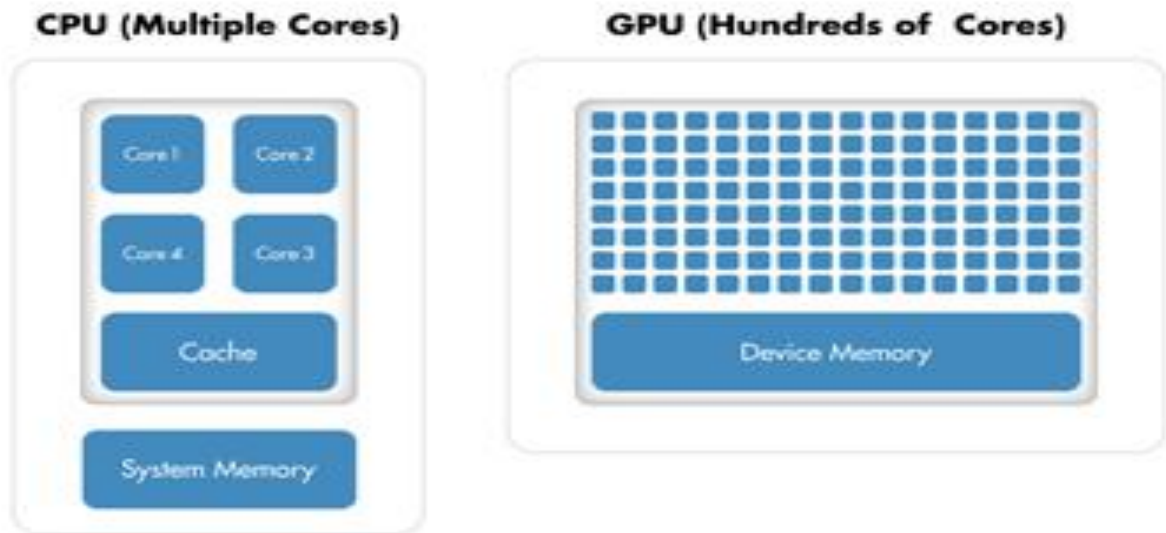


Figure 5 : Comparison of the number of cores on a CPU system and a GPU

The extraordinarily expanded throughput made conceivable by a GPU, notwithstanding, has a go at an expense.

Initially, memory access turns into a substantially more likely bottleneck for your figuring's. Information must be sent from the CPU to the GPU before computation and after that recovered from it a short time later. Since a GPU is connected to the host CPU by means of the PCI Express transport, the memory access is slower than with a conventional CPU. This implies that your general computational speedup is constrained by the measure of information move that happens in your calculation.

Second, programming for GPUs in C or FORTRAN obliges an alternate mental model and an ability set that can be troublesome and tedious to secure. Moreover, you must invest energy adjusting your code for your particular GPU to upgrade your applications for top execution.

Measuring GPU Performance

This case demonstrates to gauge a percentage of the key execution qualities of a GPU.

GPUs can be utilized to accelerate certain sorts of reckonings. Nonetheless, GPU execution changes generally between diverse GPU gadgets. To evaluate the execution of a GPU, three tests are utilized:

How rapidly can information be sent to the GPU or read back from it?

How quick can the GPU part read and compose information?

How quick can the GPU perform reckonings?

In the wake of measuring these, the execution of the GPU can be contrasted with the host CPU.

This gives an aide in respect to the amount of information or calculation is needed for the GPU to give preference over the CPU.

It is simplest to begin changing over one's code utilizing MATLAB assembled in capacities that bolster `gpuArray` information. These capacities take `gpuArray` inputs, perform computations on the GPU, and return `gpuArray` yields. A rundown of the MATLAB capacities that bolster `gpuArray` information is found in `Run Built-In Functions on a GPU`. By and large these capacities bolster the same contentions and information sorts as standard MATLAB capacities that are ascertained in the CPU. Any confinements in these overburden capacities for `gpuArrays` are depicted in their charge line help (e.g., `help gpuArray/qr`).

On the off chance that all the capacities that you need to utilize are bolstered on the GPU, running code on the GPU may be as straightforward as calling `gpuArray` to exchange data information to the GPU, and calling `accumulate` to recover the yield information from the GPU when wrapped up. By and large, you may need to victories your code, supplanting circled scalar operations with MATLAB framework and vector operations. While victimizing is by and large a decent practice on the CPU, it is normally basic for attaining to elite on the GPU. [4]

Specialists and researchers are effectively utilizing GPU innovation, initially expected for quickening illustrations rendering, to quicken their control particular computations. With insignificant exertion and without far reaching information of GPUs, you can now utilize the promising force of GPUs with MATLAB. `GPUArrays` and GPU-empowered MATLAB capacities help you accelerate MATLAB operations without low-level CUDA programming. To attain to speedups with the GPUs, your application must fulfill some

criteria, among them the way that sending the information between the CPU and GPU must take less time than the execution picked up by running on the GPU. On the off chance that your application fulfills these criteria, it is a decent possibility for the scope of GPU usefulness accessible with MATLAB.

On Windows® frameworks, a GPU gadget can be in one of two modes: Windows Display Driver Model (WDDM) or Tesla Compute Cluster (TCC) mode. For best execution, any gadgets utilized for figuring ought to be as a part of TCC mode.

Information in MATLAB exhibits is put away in section significant request. Accordingly, it is gainful to work along the first or segment measurement of your cluster. On the off chance that one measurement of your information is essentially more than others, you may attain to better execution on the off chance that you make that the first measurement. Correspondingly, in the event that you habitually work along a specific measurement, it is typically best to have it as the first measurement. At times, if continuous operations target distinctive measurements of a cluster, it may be advantageous to transpose or permute the show between these operations.

GPUs accomplish elite by figuring numerous outcomes in parallel. Accordingly, network and higher-dimensional exhibit operations ordinarily perform vastly improved than operations on vectors or scalars. You can attain to better execution by modifying your circles to make utilization of higher-dimensional operations. The procedure of updating circle based, scalar-arranged code to utilize MATLAB grid and vector operations is called victimization.

As a matter of course, all operations in MATLAB are performed in twofold accuracy skimming point number juggling. Notwithstanding, most operations bolster a mixed bag of information sorts, including whole number and single-exactness skimming point. Today's GPUs and CPUs regularly have much higher throughput when performing single-exactness operations, and single-accuracy gliding point information involves less memory. On the off chance that your application's precision necessities permit the utilization of single-accuracy coasting point, it can significantly enhance the execution of your MATLAB code.

The most ideal approach to gauge execution on the GPU is to utilize GPU time it. This capacity takes as information a capacity handle with no data contentions, and returns the deliberate execution time of that capacity. It deals with such benchmarking contemplations as rehashing the timed operation to show signs of improvement determination, executing the capacity before estimation to keep away from introduction overhead, and subtracting out the overhead of the timing capacity. Likewise, `gputimeit` guarantees that all operations on the GPU have finished before the last timing.

Case in point, consider measuring the time taken to register the lu factorization of an irregular grid A_n of size N-by-N. You can do this by characterizing a capacity that does the lu factorization and passing the capacity handle to `gputimeit`:

```
A = rand(N,'gpuArray');  
fh = @() lu(A);  
gputimeit(fh,2);
```

5. ALGORITHM

5.1 SMITH WATERMAN

Developed by T.F Smith and M.S. Waterman, the Smith-Waterman is a database search algorithm works on dynamic programming technique. It determines if the optimal alignments can be found by taking input alignments of any length, in any sequence and at any location. Depending on the calculations, scores are assigned. The exact matches are assigned positive scores and for insertions or deletions negative is assigned. Scores are added in the weight matrices and the alignment with highest score is reported. Because its ability to search a large field of possibility, it's considered exclusive than BLAST and FASTA. However there is significant decrease in the individual year-wise compression between letters. [5] Smith Waterman determine similar areas between two string or nucleotide or protein sequence. This algorithm very fast and widely use. Its last platform of sequence similarity search performance with near algorithm

5.1.1 FASTA Format

FASTA is a text base format, its exemplify nucleotide sequence or protein sequence. Its use for DNA. It had single-length code which is represented by nucleotide and amino acid. This sequence start by "<", its identifier of the sequence. In this format less than 80 character in text. Lower-case letter are accepted and mapped into uppercase. In this format not allow any numeric letter but use database to include the protein in the sequence. [6]

```
Ex- >MCHU - Calmodulin - Human, rabbit, bovine, rat, and chicken
      ADQLTEEQIAEFKFAFSLFDKDGDTITTKELGTVMRSLGQNPTEAELQDMINEVDADGNGTID
      FPEFLTMMARKMKDSTDSEEEIREAFRVFDKDGNGYISAAELRHVMTNLGEKLTDEEVDEMIREA
      DIDGDGQVNYEEFVQMMTAK [6]
```

5.1.2 BLAST (Basic Local Alignment Search Tool)

This tool for comparing primary biological sequence information. This tool search develop to compare a query sequence with a library or database of sequence and identify library sequence, and generate score like.

Ex- if query is PQGEFG

So, divide this query in different part –PQG, QGE, GEF, EFG

Score technique- if query match = +5

Unlatch = -4

So, if database sequence is PEG, PQA

1. PQG & PEG

P = 16, Q = 17,

G = 7, E = 5,

Score is = (+ (P is match)) + (- (E, Q is unmatched)) + (+ (G is match))

$$= (16+5) + (- (17+5)-4) + (7+5)$$

$$= 21 - 18 + 12$$

$$= 15$$

Similarly PQG and PQA

P = 16, Q = 17,

G = 7, A = 1,

$$= ((16)+5) + ((17)+5) + (- (7+1)-4)$$

$$= 21 + 22 + 4$$

$$= 47$$

In real meaning, we try to convert a string into another by the application some operations on single individual characters that form the string....the two possible operations can be performed, insertion or deletion of a character in the first string and the replacement of a character in the first string with the character from the second string. Thus the operations of insertion and deletion are mutually dependent, one leading to other. An edit distance is defined as the minimum possible number of operations required to convert one string into another. So, value of the alignment is defined as the similarity between two strings that maximizes the total alignment value. Then how is alignment figured out? global alignment is the result of insertion of dashes or spaces into or at the ends of two strings , and then placement of the resultant strings in a stack manner i.e one above the other so that each character or space in any of the string is opposite a unique character or a unique space in different string.

Algorithm-

$$M*N = A_{ij}$$

When

M,N is length of two sequence

i^{th} → amino acid in the first protein .
 j^{th} → one in the second protein.

Condition for any point

$$i > i_0, j > j_0$$

formula:-

$$H_{ij} = \max \left\{ \begin{array}{l} 0 \\ E_{ij} \\ F_{ij} \\ H_{i-1,j-1} + A_{ij} \end{array} \right\}$$

H_{ij} = processed matrix element

$H_{i-1,j-1}$ = score of best alignment terminated

E_{ij} = gap of column

F_{ij} = gap of row

$$E_{ij} = \max \left\{ \begin{array}{l} E_{i,j-1} - G_{\text{ext}} \\ H_{i,j-1} - G_{\text{open}} \end{array} \right\}$$

$$F_{ij} = \max \left\{ \begin{array}{l} F_{i-1,j} - G_{\text{ext}} \\ H_{i-1,j} - G_{\text{open}} \end{array} \right\}$$

G_{ext} = penalty for extending

G_{open} = penalty for opening gap

REVIEW OF LITERATURE

Ligowaski, Lukaz et al In this paper developer use the NVIDIA GeForce 9800 GX2 Dual core card and performed using CUDA lib on NVIDIA and using C++ language. They apply **smith waterman algorithm** for sequence similarity searches and alignment of similar sequence. Developer “copy the required data from global memory to shared memory, execute as many operation as possible using shared memory store the use result to global memory” this method use the developer. Developer’s kernel has 16 blocks in each block has 253 threads, so each threads processes a single sequence. So total threads is $16 \times 253=4048$. At a time kernel sequencing search is 4048/query. They test on SwissProt sequence database. This database from Swiss institute of Bioinformatics. Database succession length is somewhat under 1000 amino acids. This point of confinement of the 'evidence of idea' turning the code, which will be uprooted in further improvement. The subset of Swiss Port hold 388517 proteins (124041327 deposits 85% of whole database length) the database was change as indicated by size. Thirty seven arrangement were arbitrarily picked from the database and utilized as an inquiry. Past execution measured on the perfect case manufactured database was 8.67 GCUPS. At the point when designer perform this calculation his own framework. So his outcome is 12 GCPUS. Where all database arrangement were indistinguishable was more than 72% of that hypothetical cutoff. [7]

hasan, laiq. kentie, marijn,zaid , Al-arsIn et al this paper developer use Intel core 2 Quad Q6600, 2.4 GHz processor, with 4 GB RAM, and NVIDIA GTX275, 1GB graphics card. Developer use pre-converts the database of protien to a support format of GPU. Similarly, other Graphic Processing Unit implementation, the time intensive framework fill venture of the Smith Waterman calculation is satisfied and accelerate on the GPU. NVIDIA CUDA is utilized for DC programming (Device Code) in join with C++ for the (PC) programing (host code). In this paper designer utilization Swiss-Port database for examination. This database is larger than 1GB to divide in small part before using loaded into the GPU global memory for alignment. In execution smith waterman give back the most extreme scores embedded of the real arrangement, and skirting the calculation follow back step meaning

improves and paces up the usage. In this grouping document can then be adjusted on host PC utilizing the smith- waterman pursuit instrument. By default 20 top-scoring sequencing are excess, while the Swiss-Port database contains more than 500,000. Developer convert the data format is FASTA to GPU format to better much compatibility. In help of this research developer increase the performance around 21.4 GCUPS (Graphical Cell Update Per Second). [8]

eng Xiaowen, jin, hai, zheng, rain, zhu, lei, dai, weiqi et al In this paper developer use the CUDA architecture. When a kernel grid is invoked by CUDA program the grid block are enumerated and distributed to smith waterman. The execution of threads belonging to threads blocks takes place. It's required a huge amount of computation and memory space, and is also constrained by memory access speed of GPU , when accelerate global memory using GPU. Developer design to provide the best alignment of all the database sequence and they decrease the number of the memory access to reduce memory bandwidth bottleneck. Developer have some challenges, they improve the organize data and placement strategies in GPU memory hierarchy. On-chip shared memory use the better utilization by low-latency. Load balancing amount of threads. In this paper developer use SapceAlign for mapping because smith waterman algorithm is so difficult to map, so GPU implement new implementation of smith waterman alignment. Its improve performance with several solution, increasing GPU threads for each database sequence. In the database data is very large and unbalance, so its sequence are length of data base are various, and resulting load imbalance in GPU. Alleviate load imbalance in a warp to some extent, however there is still high load imbalance among warps. In this paper developer's process can be helpful to other application which is less amount of input and output, but large amount of intermediate result, such as Viterbi path scanning. Combination between them increase the performance of application. [9]

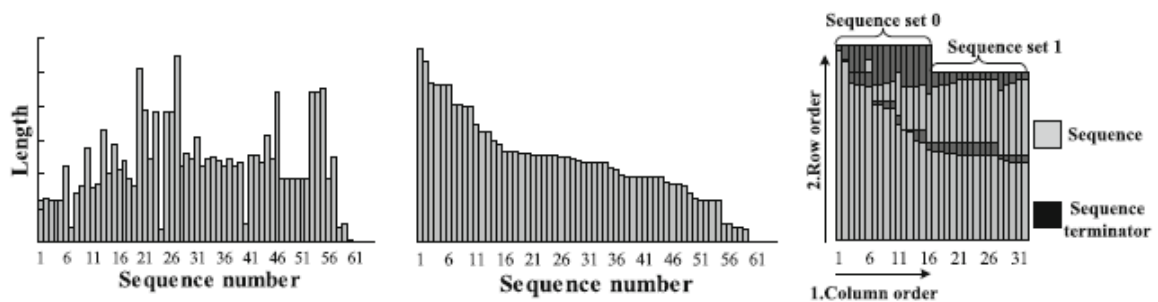


Figure 6 An example of database sequences conversion. **a** Original database sequences, **b** sorted database

Panagiotis, vouzis, nikolaos , shainidis et al In this paper developer, develop the GPU-BLAST for accelerated version of the popular NCBI-BLAST. BLAST (Basic Local Alignment Search Tool) is Bio-informatics tool, which is widely used. This tool use for the two different biological sequences comparison. this implementation based on Source Code of NCBI-BLAST, GPU-BLAST and NCBI-BLAST input output result are same. Developer share the many data structure with NCBI-BLAST. They modified NCBI-BLAST with GPU but without compromising the accuracy. When they use single thread on GPU-BLAST then it work 4 time faster, and they use six thread it works twice as faster. They use the smith waterman algorithm, in this algorithm they use three main step seeding, extension, and evaluation. In the seeding, two different types of sequence marching , its identifies short word common between the query and a database sequence and user them as seeds. Second is exertion in this step discards the false positive seeds that generate by chance and takes the seeds that generate because there are part of longer common sequence. Third is evaluation, in this step generate the score produced by the ungapped or gapped extension step, the query and database sequence lengths, the sequence data. Using of six CPU threads that run parallel with GPU outputs the average GPU-BLAST speedup. The elapsed times are used for sequence alignment with former starting from GPU blast execution and ending with writing the output alignments to a file. When compared single threaded NCBI-BLAST the GPU-BLAST achieves the largest speed ups and they fall with increase in the CPU threads. Both CPU/GPU and multi-threaded CPU tend to remain a function of many available CPU threads. In every case the total time used to align the whole set of queries was used to calculate the speedups. there is no linear scaling in case of multi-threaded NCBI-blast. With six threads the NCBI-BLAST speedup remains four. GPU-BLAST acquires some limitations as it is directed towards getting the same output results for which it is built on NCBI-BLAST. However, in every case the addition of the GPU significantly boosts the observed speedups. For instance, for both gapped and ungapped alignments the CPU-BLAST is able to achieve a speedup of nearly six. [10]

razmyslovich, Dzmitry; Marcus, Guillermo; Gipp, Markus; Marc, Zapatka; Szillus, Andreas et al In this implementation developer use NVIDIA GeForce GTX260 GPU, 1.75 GB RAM, Intel i7-920 CPU, 6GB RAM on Linux OS with NVIDIA GPU SDK3.0. In this paper researcher use the smith waterman algorithm. They implement for sequence path calculation and computing similarity index between query sequence and references. This implementation application for cancer research. This implementation is 3 time faster

the CUDA-enabled CUDASW++2.0 for large and medium sequences. Their OpenCL model programming consists of two parts. First is the help of Host code to load all data into GPU memory and schedule a kernel execution and its result. Second is the kernel code executed on GPU. In implementation they use some steps-

- (1) Parallelization granularity- all the data stored in parallel according to the execution of data sequencing.
- (2) Long reference sequence processing- they use large reference sequences divided into two parts.
 - (i) New piece of the matrix calculation.
 - (ii) New optimal path together with truncating the current matrix and calculating it.
- (3) Multi-query processing- In CPU one query is processed at a time on one multiprocessor. In the help of GPU, several queries are processed at a time.
- (4) The calculating shape- this result calculates a model to make a shape it important to choose the right shape.
- (5) The concurrent transfer and execution- In this step Host initialization and transfer the data to other device memory provide the kernel scheduling path calculation.
- (6) Smith waterman without the path calculation- Input the data in matrix form and execute the algorithm.

They increase the performance compared to CPU 9x with path and 130x without path. It is also 3x faster than CUDASW++ . [11]

Edans Flavius de O. Sandes and Alba Cristina M.A. de Melo, Senior Member et al In this research researcher was using the NVIDIA GTX Ti 1GB graphics card. They use CUDA 2.1 its predefined program version. They use the SRA database which is 50 GB genomic sequence. In this paper researcher are divide the algorithm in six parts. In first part the all processes in DP matrix, one is Special rows it's a saved area. In Next part DP matrix fill the like back tracking direction end point to starting point, and this processes saved in another special columns in disk. This optimization is also called the orthogonal exe. This calculation are reduces the part 2. And then part execution is work similarly part 2 and in this part number of cross points in executions. After then part 3 they decrease the size of partitions with the help of Orthogonal and MM algorithm, in the help of this part they reduce the size of all partition size. Next part 5 is finding the string marching on all partition. The next part 6 is optional according to researcher they represent the all result

and graphical result. They completed all 50 GB database sequence alignment within 8 hr. 29 min [12]

Yongchao Liu and Bertil Schmidt et al In this paper researcher use CUSHAW2–GPU, a CUDA good gapped short-read aligner in view of CUSHAW2 calculation. In this aligner, an entomb undertaking cross use CPU–GPU parallelism has been explored to simultaneously adjust distinctive peruses on both multicore CPUs and GPUs. What's more, another tiled-based SW arrangement backtracking calculation has been formulated utilizing CUDA to encourage quick arrangement. CUSHAW2–GPU acknowledges FASTA, FASTQ, SAM, and Binary arrangement Alignment/Map designs as data, and reports arrangements in SAM position. They evaluated the execution of CUSHAW2–GPU and different aligners, utilizing recreated and genuine peruses. On re-enacted information, CUSHAW2–GPU yields better review and accuracy than BWA–SW, Bowtie2, and GEM for both SE and PE arrangements. On genuine information, CUSHAW2–GPU has exhibited equivalent or better affectability. Concerning speed, our aligner with a NVIDIA Tesla K20c GPU can attain to a speedup of up to 2.8, 4.2, 1.7, and 2.0 in examination to the multithreaded CUSHAW2, BWA–SW, Bowtie2, and GEM on the 12 centres of a top of the line CPU for the SE arrangement, and up to 1.6, 2.7, 1.3, and 2.1 for the alignment [13]

Jung-Hyun Hong, Young-Ho Ahn, Byung-Jin Kim and Ki-Seok Chung et al In this paper researcher made new frame work for OpenCL this framework will provide advantages for other programing frameworks, this framework mainly deign for hybrid performance with CPU and GPUs. In the help of this framework provide multi-core for CPU, and also provide the boost up the performance. This platform support for host-kernel program environment and different model like NDK, SDK for mobile and LLVM. In this OpenCL framework use the kernel command query to arrange and also manage number of threads and handle. They implement some additional function for OpenCL it's called OpenCL kernel. The purpose of researcher implement of function is provide the increases the parameter, multiple design increases the size of work-group and item. We know that previously in c not use parallel computing in the help of OpenCL we use the parallel computing. With the help of OpenCL we improve the performance, handle the memory device on C. in this paper researcher are explain how to use C in OpenCL framework. [14]

Romain Dolbeau, Francis Bodin, Guillaume Colin de Verdière et al. in this paper researchers say OpenCL gives a convenient API to advantage from quickening agent innovation. All things considered, code tuning remains an issue when sending a solitary code base on diverse stages. If one form of the code is to be utilized for all equipment, they demonstrate that it is vital to search for a tradeoff between normal productivity and best execution. For the situation uncovered in this paper they exhibit that execution convey ability is achievable if an effectiveness loss of 12% is considered as adequate. Variants of the code tuned for maximal effectiveness for a given equipment (h/w) have a tendency to firmly corrupt execution on alternate frameworks (up to 43% loss of throughput) and may not even run. This paper demonstrates that, if extreme execution is not the objective, execution movability can be accomplished to a degree. As likewise indicated in past studies, when searching for exchange offs, auto-tuning innovation is an unquestionable requirement. This work shows that it is not generally important to install self-adjusting codes that prompts runtime overheads and additional unpredictability. This study will be augmented utilizing more applications as a part of request to better focus when self-adjusting code is entirely vital. [15]

T. Brandes; A. Arnold; T. Soddemann; D. Reith et al In this paper specialist say what are the distinction in the middle of GPU and CPU .In both cases, GPUs and CPUs, expanding maximal execution obliges mindful tuning to fit the store and memory orders. On CPUs, the key is capable ill-use of the broad stores by fitting blocking. On GPUs, one needs to evaluate unmistakable memory access outlines due to different strolls and figure capacities. Great heuristics for picking perfect access illustrations are basic for execution streamlining. BLAS timetables are a sensible include as they give extraordinary, but not perfect execution both on CPUs and GPUs and a fundamental utilization. On CPUs this is particularly valid for C programs, which the explored compiler families enhance a great deal not exactly FORTRAN codes. On GPUs, the BLAS-2 schedules have enhanced and are new tantamount to very much streamlined hand-tuned code, albeit just on late equipment. [16]

Youquan Liu; Shaohui Jiao; Wen Wu; Suvranu De et al The paper presents a new solution having the capability of high speed dynamic deformation simulation on the most recent available graphics processing unit hardware with CUDA which is from NVIDIA. CUDA based GPUs generate the power of about 128 microprocessors allowing parallel data computations. The availability of c language interface makes it more flexible than the

previous available GPGPU. The arrangement is given in type of limited component system. The test outcomes got from the experimentation demonstrate that GPU with CUDA improves to four times in velocity up for limited component strategy deformity calculation on Intel(R) Center 2 Quad 2.0GHz machine with GeForce 8800 GTX. [17]

John D.; Mike Houston; David Luebke et al The paper depicts the equipment, foundation and programming model for graphical preparing unit. It clarifies the working of instruments and strategies in GPU and the paper displays around four examples of GPU processing victories on diversion physical science and computational biophysics that give higher execution increases over effectively advanced CPU applications. It satisfies its truths on the premise of the headway that has taken in GPU registering throughout the years. The present GPU is currently an effective representation motor as well as an exceedingly programmable processor. [18]

Ignacio, Juan; Juan Ignacio Perez; García, Eliseo; Frutos, José A. de et al The paper utilizes attributes premise capacity strategy to run on the representation handling unit. CBFM is high parallel procedure which makes itself accessible for abusing the parallel assets of GPU. The rates up over 90 are gotten and these outcomes appear to be very encouraging. The outcomes can be streamlined with cautious hand-tuning and enhancement. This thusly has a tendency to open numerous conceivable outcomes like expanding the size and precision of EM investigation to performing on normal workstation. [19]

Changyou Zhang,; Kun Huang; Xiang Cui; Yifeng Chen et al The paper contemplates a mapping from equipment to primitives to help abnormal state programs mindful of the exchange off in the middle of execution and force utilization on GPU quickening agent. The paper introduces a model calling attention to an arrangement of primitives which change the string state. A string state change is moderately simple for developer to be mindful of and the abnormal state dialect measures the relating force change. The consequences of the force utilization estimations of specific primitives and with a given information exchanging proficiency outline to help software engineers to be knowing the force effectiveness of the fundamental code. [20]

Mingcheng Wu,; Jingyi Fu; Xiaorong Hu et al The paper on basis of GPU computing presents a relatively fast multi-GPU that comes with the capability of aligning large amount of T-receptor nucleotide sequences. A half and half of CPU and multi-GPU is proposed utilizing CUDA –enabled Fermi GPU and the NVIDIA gave CUDA tool stash. This component is a configuration for quicker and considerably more viable arrangement process. The two i.e. CPU and GPU are diverse in working, CPU being in charge of rationale control and GPU for parallel figuring. Between assignment parallel methodology when connected in segment of parallel registering is in charge of bringing high parallelism as well as not makes the arrangement procedure limited to specific arrangement calculation. Multi-GPU, single GPU and single CPU registering are utilized under the basic equipment for the arrangement of mouse TCR nucleotide groupings. The outcomes demonstrated the multi-GPU as the best of all in execution and cost. [21]

Michael J. Dinneen,; Masoud Khosravani; Andrew Probert et al The paper presents a way to use the power of multiple GPUs in form of two design techniques. The two techniques being using a host CPU script to synchronize a distributed view of a graph algorithm where in every node of the input graph is related with a unique processing thread ID and using GPU simplest atomic operations to synchronize a single kernel launch where in a set of threads, upper-bounded by at most the number of streaming processing units available, stay active at all time intervals and divide the total workload until the algorithm completes. A reliable comparative work of both the techniques is presented using BFS. The paper concludes on the fact that OpenCL and CUDA both are natural tools available for graph algorithm designers especially for those who are not the experts of GPU hardware architecture. This helps developers to develop real-world usable graph applications. [22]

Ching-Lung Su, Po-Yu Chen, Chun-Chieh Lan, Long-Sheng Huang, and Kuo-Hsuan Wu et al In this paper, we utilized a lot of comparable portions to think about the registering execution of C, OpenCL and CUDA, the two sorts of API's on NVIDIA Quadro 4000 GPU. The exploratory result demonstrated that, the official time of CUDA Driver API was 94.9%~99.0% speedier than that of C, while and the official time of CUDA Driver API was 3.8%~5.4% speedier than that of OpenCL. Likewise, the cross-stage normal for OpenCL did not influence the execution of GPU. In this paper we connected five application benchmarks to look at the adequacy of C, OpenCL and CUDA, the two sorts of

API's. The correlation and examination result indicated that the official time of CUDA Driver API was 94.9%~99.0% speedier than that of C, while the official time of CUDA Driver API was 3.8%~5.4% speedier than that of OpenCL. Likewise, the additional official time needed by the between stage normal for OpenCL, did not appear influence the general execution of OpenCL. Additionally, there was little contrast between the system configuration of OpenCL and CUDA. This permits specialists and designers to pick one sort of programming model to control GPU as indicated by their need, and accordingly upgrade the general framework viability and decrease advancement time. [23]

Jianbin Fang, Ana Lucia Varbanescu and Henk Sips et al This paper exhibits a thorough execution correlation in the middle of CUDA and OpenCL. We have chosen 16 benchmarks going from engineered applications to certifiable ones. We make a broad examination of the execution crevices considering programming models, advancement techniques, building points of interest, and hidden compilers. Our outcomes show that, for most applications, CUDA performs at most 30% better than OpenCL. We likewise demonstrate that this distinction is because of unreasonable examinations: actually, OpenCL can attain to comparable execution to CUDA under a reasonable examination. In this way, we characterize a reasonable examination of the two sorts of uses, giving rules for more potential investigations. We additionally examine OpenCL's portability by running the benchmarks on other winning stages with minor changes. In general, we reason that OpenCL's versatility does not on a very basic level influence its execution, and OpenCL can be a decent different option for CUDA. Since it has been demonstrated in this paper that OpenCL is a decent different option for CUDA, we might want to build up an auto-tuner to adjust broadly useful OpenCL projects to all accessible particular stages to completely misuse the equipment. [24]

Baida Zhang Shuai Xu Feng ZhangYuan Bi and Linqi Huang et al In this paper the examination is too a structure for system consequently chooses which work is expense productive to execute on GPU. A progression of benchmark of distinctive sorts of registering, including information exchange between GPU and CPU, information network Generation, grid operation and GPU capacities were tried in each of the three tool stash. Furthermore, the outcomes demonstrate that Jacket is the best one. A few advices to enhance the execution of tool stash are given at last . An examination is given for when an errand ought to be executed on GPU. Also, a few measures are gotten from the work. A

execution assessment of three MATLAB GPU tool kits is too displayed here. Also, Jacket is the champion in numerous perspectives. It has the best execution and backings the most capacities than other 2 tool kits. Also, there are additionally a few issues exist. The client not just need to know a GPU exist in his PC, yet additionally need to know which MATLAB capacity the tool stash support. Also, when he composes script, he must to PC what number of times the information exchange spent, this is an overwhelming weight. The best way is the client require not to know considerably whether a GPU exist, much the same as we need not to know how CPU functions. The tool stash will pick suitable capacity and run it on GPU. We are presently doing the work about naturally investigation and timetable a few capacities to execute on GPU and the client need not to know there is a GPU exist. This is the thing that the general client needed. [25]

PRESENT WORK

3.1 Problem Formulation

Today as we are rising far ahead in technologies organizations require best performance systems which could perform hardest calculations efficiently and effectively these calculations require a large amount of processing and for processing we require CPU but using CPU we could execute only calculations in series due to this the amount of time consumed is more. Here comes demand of GPU which execute calculations in Parallel. Researchers has tried to provide results by performing calculations using different systems with different capabilities working with OpenCL an open source platform which supports all types of system and CUDA only for NVIDIA GPUs .we are trying to create an environment where we are using **these two platform in single system** and comparing the results for them. We found that it was the first time smith waterman algorithm is used on GPU using MATLAB where we see MATLAB has limited functionality with CUDA the NVIDIA GPU.

OBJECTIVE

The objective of this research is accelerate smith waterman algorithm with the help of OpenCL.

- To implement smith waterman algorithm on CUDA and OpenCL.
- To compare the performance of smith waterman algorithm using OpenCL with visual studio and CUDA with MATLAB.
- To find out the parameter which effect the performance of this smith waterman algorithm on OpenCL and CUDA.
- Check the amount of acceleration of smith waterman algorithm using OpenCL.

RESEARCH METHODOLOGY

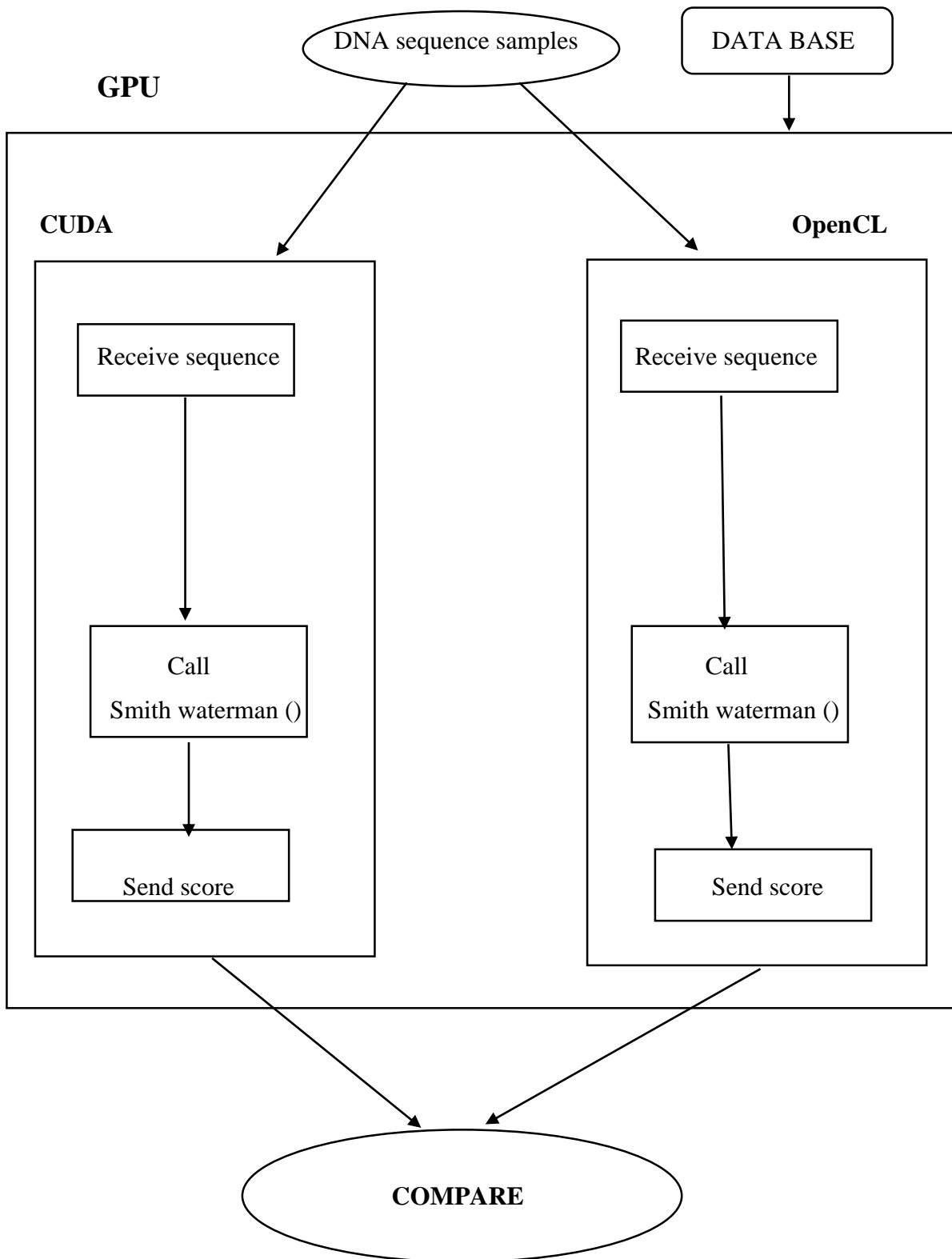


Figure 7: Flow diagram

Step 1: Pick any DNA sequence randomly.

-In this step I will pick any DNA sequence randomly. Which is FASTA format.

Step 2: Use Two techniques OpenCL and CUDA.

-In this step in my GPU I will use two techniques OpenCL and CUDA. Both techniques perform operation one by one.

Step 3: Use DNA database.

-In this step I will download use any sequence database like NCBI or Uni-Prot sequence Database, this database freely provided for researchers. Then Database connected to smith waterman algorithm.

Step 4: Use CUDA technique

In this step in the help of C++ language access DNA sequence in CUDA.

Step 5: Use Smith Waterman in CUDA.

In this step I will create smith waterman program in C++. Then CUDAs kernel call to smith program.

Step 6: Generate Score.

In this step Smith waterman algorithm DNA sequence match with database and generate Score.

Step 7: Note result.

In this step I will note all the result like execution time, how many use threads, how many use cores.

Step 8: Use OpenCL technique.

In this step use step 4 to 7 on OpenCL and Note all the result.

Step 9: Compare

In this step I will compare all the result it's time, core and threads.

Step 10: Accelerate OpenCL

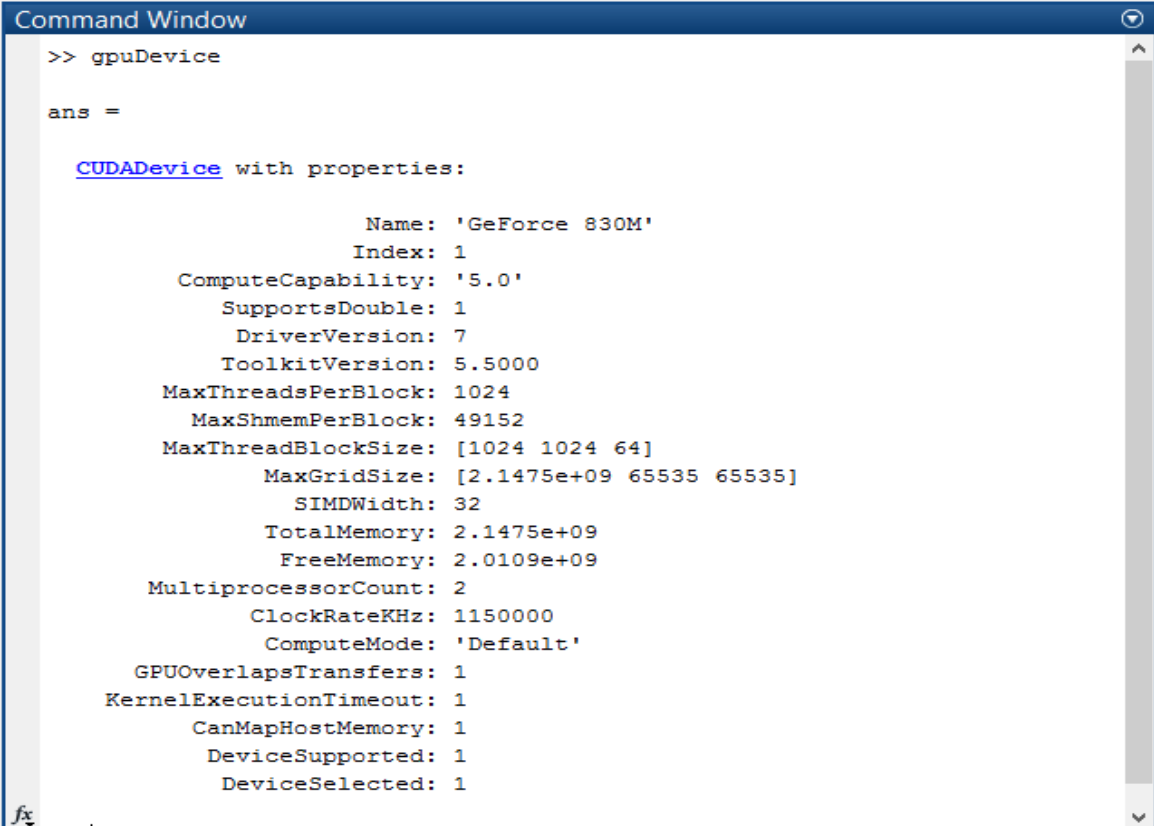
In this step after analysis all result I will try to accelerate OpenCL result.

RESULTS AND DISCUSSION

Using MATLAB we have found results of comparing two strings on CUDA the NVIDIA GPU platform. We have use two strings one of which is DNA sequence of Human Being and other is a DNA sequence of H1N1 virus infected Human Being. After executing of the smith waterman algorithm we got the similarity score for the two strings entered.

6.1 Gathering Required Data

1. To show GPU property using MATLAB command `gpuDevice`



```
Command Window
>> gpuDevice

ans =

  CUDADevice with properties:

      Name: 'GeForce 830M'
      Index: 1
  ComputeCapability: '5.0'
    SupportsDouble: 1
      DriverVersion: 7
    ToolkitVersion: 5.5000
MaxThreadsPerBlock: 1024
  MaxShmemPerBlock: 49152
MaxThreadBlockSize: [1024 1024 64]
      MaxGridSize: [2.1475e+09 65535 65535]
      SIMDWidth: 32
      TotalMemory: 2.1475e+09
      FreeMemory: 2.0109e+09
MultiprocessorCount: 2
      ClockRateKHz: 1150000
      ComputeMode: 'Default'
GPUOverlapsTransfers: 1
KernelExecutionTimeout: 1
  CanMapHostMemory: 1
  DeviceSupported: 1
  DeviceSelected: 1
```

Figure 8: Property of GPU

- To find the number of cores in GPU “NVIDIA GeForce 830M”.

```

Command Window
Name: GeForce 830M
Index: 1
ComputeCapability: '5.0'
SupportsDouble: 1
DriverVersion: 7
ToolkitVersion: 5.5000
MaxThreadsPerBlock: 1024
MaxShmemPerBlock: 49152
MaxThreadBlockSize: [1024 1024 64]
MaxGridSize: [2.1475e+09 65535 65535]
SIMDWidth: 32
TotalMemory: 2.1475e+09
FreeMemory: 2.0109e+09
MultiprocessorCount: 2
ClockRateKHz: 1150000
ComputeMode: 'Default'
GPUOverlapsTransfers: 1
KernelExecutionTimeout: 1
CanMapHostMemory: 1
DeviceSupported: 1
DeviceSelected: 1

>> gpuCores = gpu1.MultiprocessorCount * gpu1.SIMDWidth

gpuCores =

    64

fx >>

```

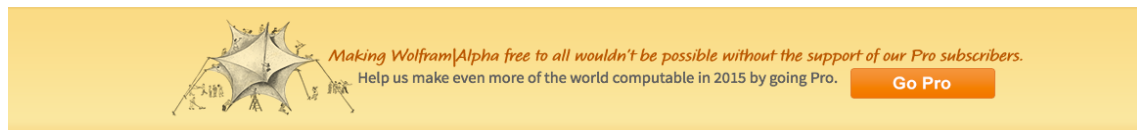
Figure 9: No of Cores in GPU

- Taken the H1N1 virus genome sample. Downloaded by NCBI web site.

The screenshot shows the NCBI GenBank entry for the H1N1 swine influenza virus genome assembly A/England/251/2011, chromosome 8. The page includes a search bar, display settings (FASTA), and a list of external resources like NEP cDNA clones and NS2/NS1 cDNA clones.

Figure 10: H1N1 Sample

4. Human being DNA sample downloaded from www.WolframAlpha.com --



A screenshot of the WolframAlpha website. The search bar contains the DNA sequence "AAGCTAGCTAGC". Below the search bar, the results are displayed: "Input interpretation: AAGCTAGCTAGC (genome sequence)", "Length: 12 base pairs", "Amino acid sequence: (5'-3' frame 1) AAG | CUA | GCU | AGC | Lys | Leu | Ala | Ser", and "Oligonucleotide melting temperature: 48.5 °C (degrees Celsius)". There are also buttons for "All reading frames" and "Use single letters". On the right side, there are social media sharing options (Facebook, Twitter, LinkedIn, etc.) and a "New to Wolfram|Alpha?" section.

Figure 11: Human Being DNA sample

6.2 Analysis of Smith Waterman Algorithm

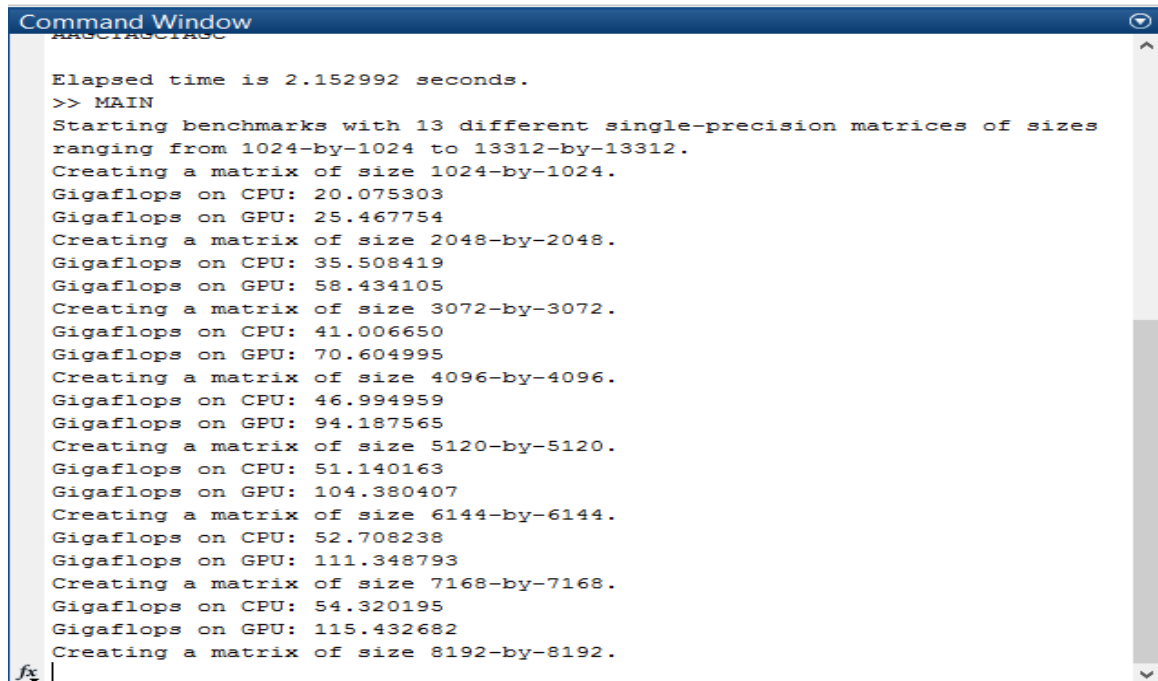
6.2.1 CUDA USING “MATLAB”

1. Two different DNA sequence.

Sequence 1 = ‘ATGGACTCCAACACCATGTC’

Sequence 1 = ‘AAGCTAGCTAGC’

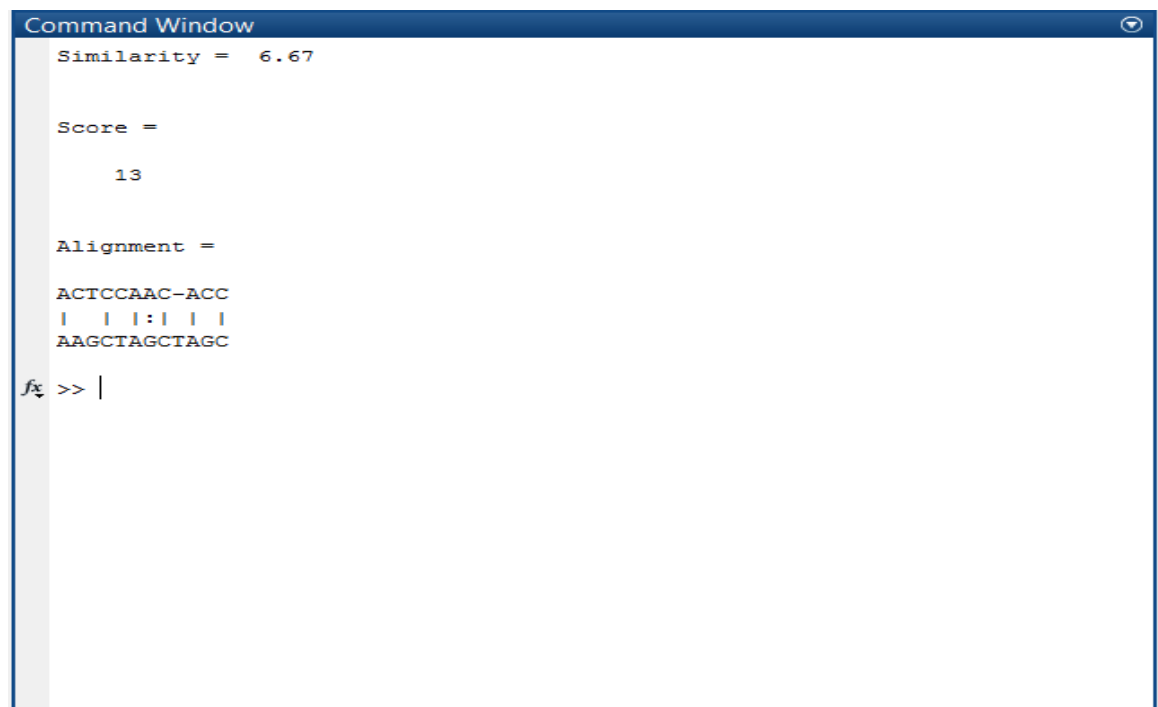
2. To show Execution and Analysis of performance using gigaflops on GPU and CPU.



```
Command Window
Elapsed time is 2.152992 seconds.
>> MAIN
Starting benchmarks with 13 different single-precision matrices of sizes
ranging from 1024-by-1024 to 13312-by-13312.
Creating a matrix of size 1024-by-1024.
Gigaflops on CPU: 20.075303
Gigaflops on GPU: 25.467754
Creating a matrix of size 2048-by-2048.
Gigaflops on CPU: 35.508419
Gigaflops on GPU: 58.434105
Creating a matrix of size 3072-by-3072.
Gigaflops on CPU: 41.006650
Gigaflops on GPU: 70.604995
Creating a matrix of size 4096-by-4096.
Gigaflops on CPU: 46.994959
Gigaflops on GPU: 94.187565
Creating a matrix of size 5120-by-5120.
Gigaflops on CPU: 51.140163
Gigaflops on GPU: 104.380407
Creating a matrix of size 6144-by-6144.
Gigaflops on CPU: 52.708238
Gigaflops on GPU: 111.348793
Creating a matrix of size 7168-by-7168.
Gigaflops on CPU: 54.320195
Gigaflops on GPU: 115.432682
Creating a matrix of size 8192-by-8192.
```

Figure 12: Results in GIGAFLOPS

3. After complete execution on MATLAB we get values similarity and score. We also get alignment between two strings.



```
Command Window
Similarity = 6.67

Score =

    13

Alignment =

ACTCCAAC-ACC
| | |:| | |
AAGCTAGCTAGC

fx >> |
```

Figure 13: Alignment and Score

4. Graphical representation for high score point in Smith waterman algorithm matrix.

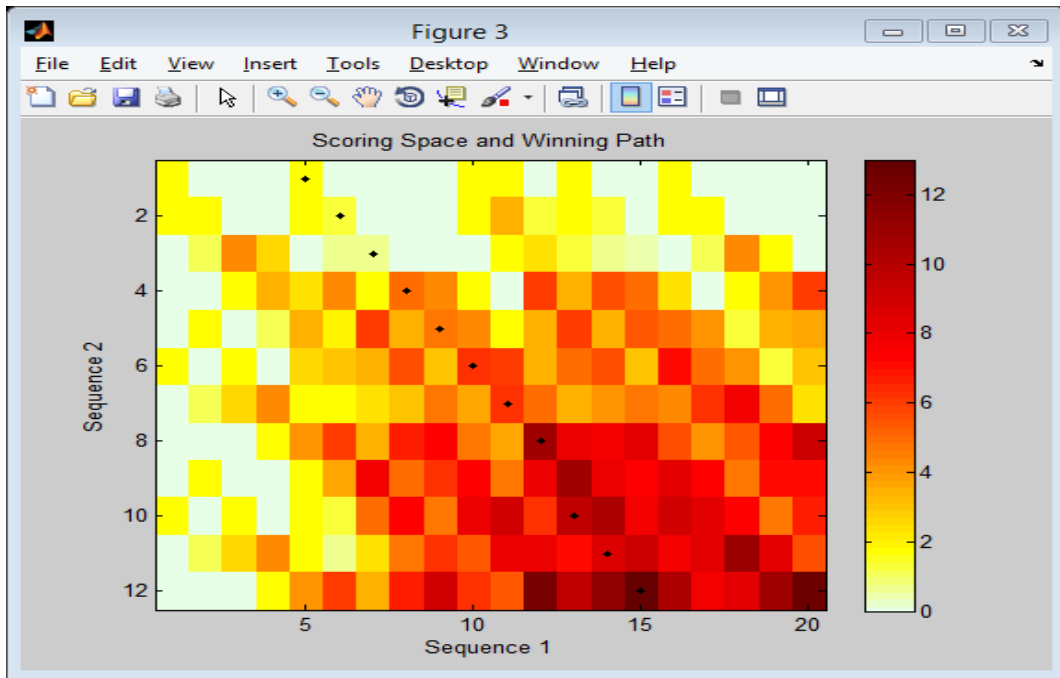


Figure 14: SW algorithm matrix

5. Benchmarking results from GPU and CPU gigaflops.

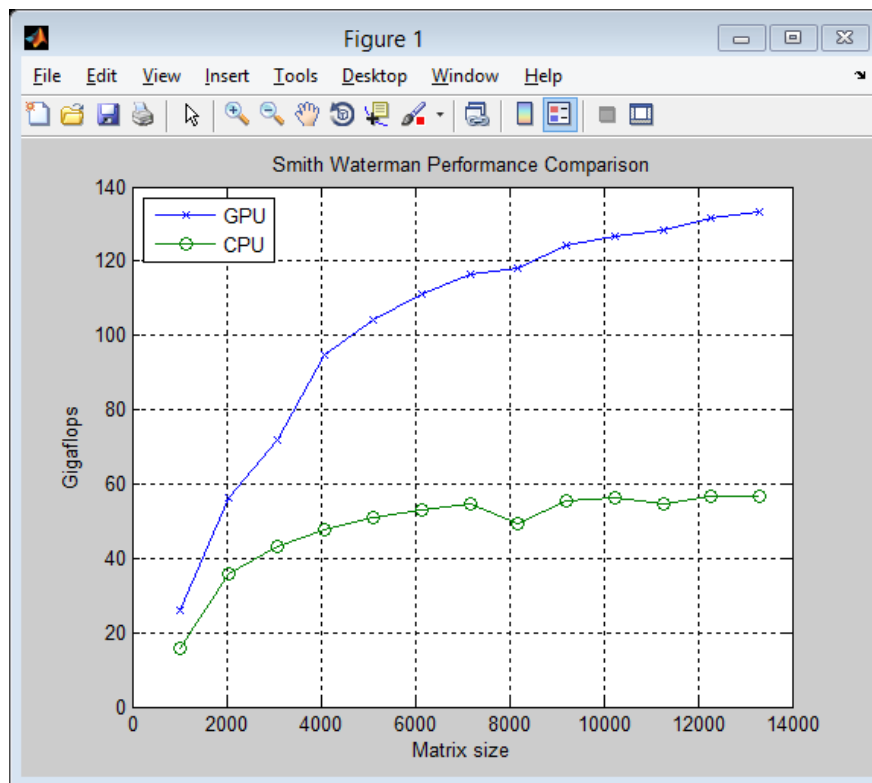


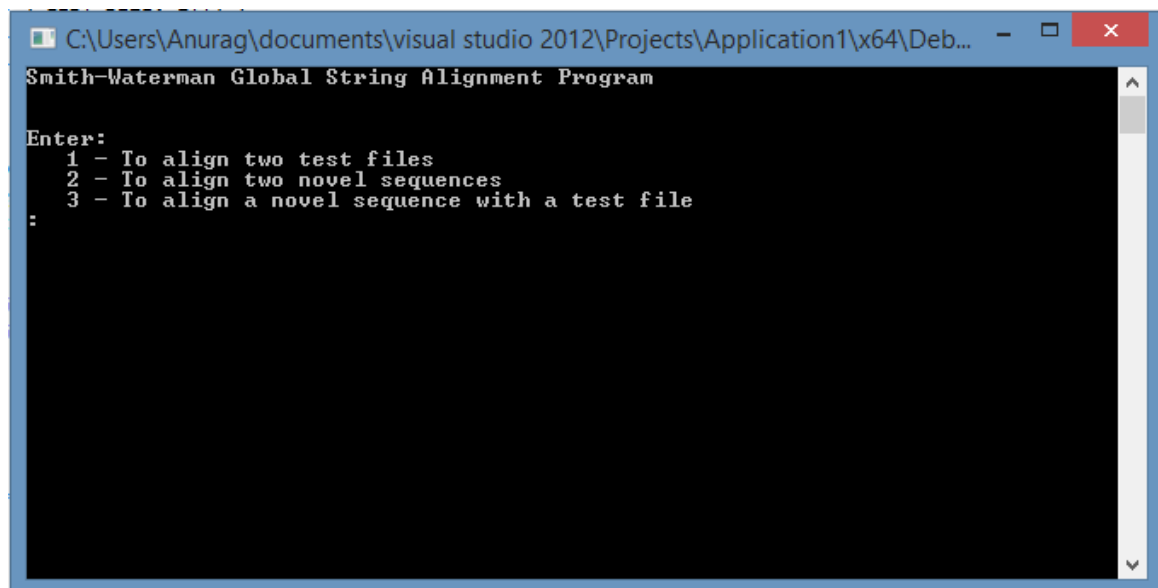
Figure 15: Benchmark Results

6.2.2 OPENCL WITH VISUAL STUDIO

Visual Studio is used to recreate smith waterman algorithm for Open CL which is an open source platform for results like comparing performance of CPU and GPU. MATLAB is used to recreate the same algorithm for CUDA which is NVIDIA GPU Platform.

We also have compare results based on smith waterman algorithm for Open CL in Visual Studio which have three selections first of which is used to find the similarity score between two fa(FASTA) files. Second selection which is used to compare two user-defined strings which gives us the similarity score calculated by comparing the sting and find sequence in the matrix generated by smith waterman algorithm. This also calculate execution time for CPU and GPU. Third selection which is used to find the similarity score by comparing a fa(FASTA) file and user-defined string.

1. Smith Waterman Algorithm in OpenCL provides three selections. For Comparison and Analysis of two files or two sequences or a sequence and a file.



```
C:\Users\Anurag\documents\visual studio 2012\Projects\Application1\x64\Deb... - [x]
Smith-Waterman Global String Alignment Program
Enter:
  1 - To align two test files
  2 - To align two novel sequences
  3 - To align a novel sequence with a test file
:
```

Figure 16: Three Selections

Selection 1: this shows how to align two test files.

In this two fa (FASTA) file are use to produce different strings of different size as example

str1.fa = DNA string of human

str2.fa = H1N1 virus DNA string

We see a matrix using smith waterman algorithm when two strings are str1 and str2 on CPU

```

C:\Users\Anurag\documents\visual studio 2012\Projects\Application1\x64\Deb...
Enter:
1 - To align two test files
2 - To align two novel sequences
3 - To align a novel sequence with a test file
:1
  0 A T G G A C T C C A A C A C C A T G T C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
A 0 5 0 0 0 5 0 0 0 0 5 5 0 5 0 0 5 0 0 0
A 0 5 1 0 0 5 1 0 0 0 5 10 1 5 1 0 5 1 0 0
G 0 0 1 6 5 0 1 0 0 0 0 1 6 0 1 0 0 1 6 0 0
C 0 0 0 0 2 1 5 0 5 5 0 0 6 2 5 6 0 0 2 5
T 0 0 5 0 0 0 0 10 0 1 1 0 0 2 0 1 2 5 0 5
A 0 5 0 1 0 5 0 0 6 0 6 6 0 5 0 0 6 0 1 0
G 0 0 1 5 6 0 1 0 0 2 0 2 2 0 1 0 0 2 5 0
C 0 0 0 0 1 2 5 0 5 5 0 0 7 0 5 6 0 0 1 5
T 0 0 5 0 0 0 0 10 0 1 1 0 0 3 0 1 2 5 0 5
A 0 5 0 1 0 5 0 0 6 0 6 6 0 5 0 0 6 0 1 0
G 0 0 1 5 6 0 1 0 0 2 0 2 2 0 1 0 0 2 5 0
C 0 0 0 0 1 2 5 0 5 5 0 0 7 0 5 6 0 0 1 5
  0 0 0 0 0 0 0 1 0 1 1 0 0 3 0 1 2 0 0 0
AA
AA Press any key to continue . . .
Execution time in milliseconds = 0.218 ms
  
```

Figure 17: Selection 1 on CPU

Selection 1 based on GPU

```

C:\Users\Anurag\documents\visual studio 2012\Projects\Application1\x64\Deb...
Enter:
1 - To align two test files
2 - To align two novel sequences
3 - To align a novel sequence with a test file
:1
  0 A T G G A C T C C A A C A C C A T G T C
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
A 0 5 0 0 0 5 0 0 0 0 5 5 0 5 0 0 5 0 0 0
A 0 5 1 0 0 5 1 0 0 0 5 10 1 5 1 0 5 1 0 0
G 0 0 1 6 5 0 1 0 0 0 0 1 6 0 1 0 0 1 6 0 0
C 0 0 0 0 2 1 5 0 5 5 0 0 6 2 5 6 0 0 2 5
T 0 0 5 0 0 0 0 10 0 1 1 0 0 2 0 1 2 5 0 5
A 0 5 0 1 0 5 0 0 6 0 6 6 0 5 0 0 6 0 1 0
G 0 0 1 5 6 0 1 0 0 2 0 2 2 0 1 0 0 2 5 0
C 0 0 0 0 1 2 5 0 5 5 0 0 7 0 5 6 0 0 1 5
T 0 0 5 0 0 0 0 10 0 1 1 0 0 3 0 1 2 5 0 5
A 0 5 0 1 0 5 0 0 6 0 6 6 0 5 0 0 6 0 1 0
G 0 0 1 5 6 0 1 0 0 2 0 2 2 0 1 0 0 2 5 0
C 0 0 0 0 1 2 5 0 5 5 0 0 7 0 5 6 0 0 1 5
  0 0 0 0 0 0 0 1 0 1 1 0 0 3 0 1 2 0 0 0
AA
AA Press any key to continue . . .
Execution time in milliseconds = 0.016 ms
  
```

Figure 18: Selection 1 on GPU

Section 1: Compare CPU and GPU time Result

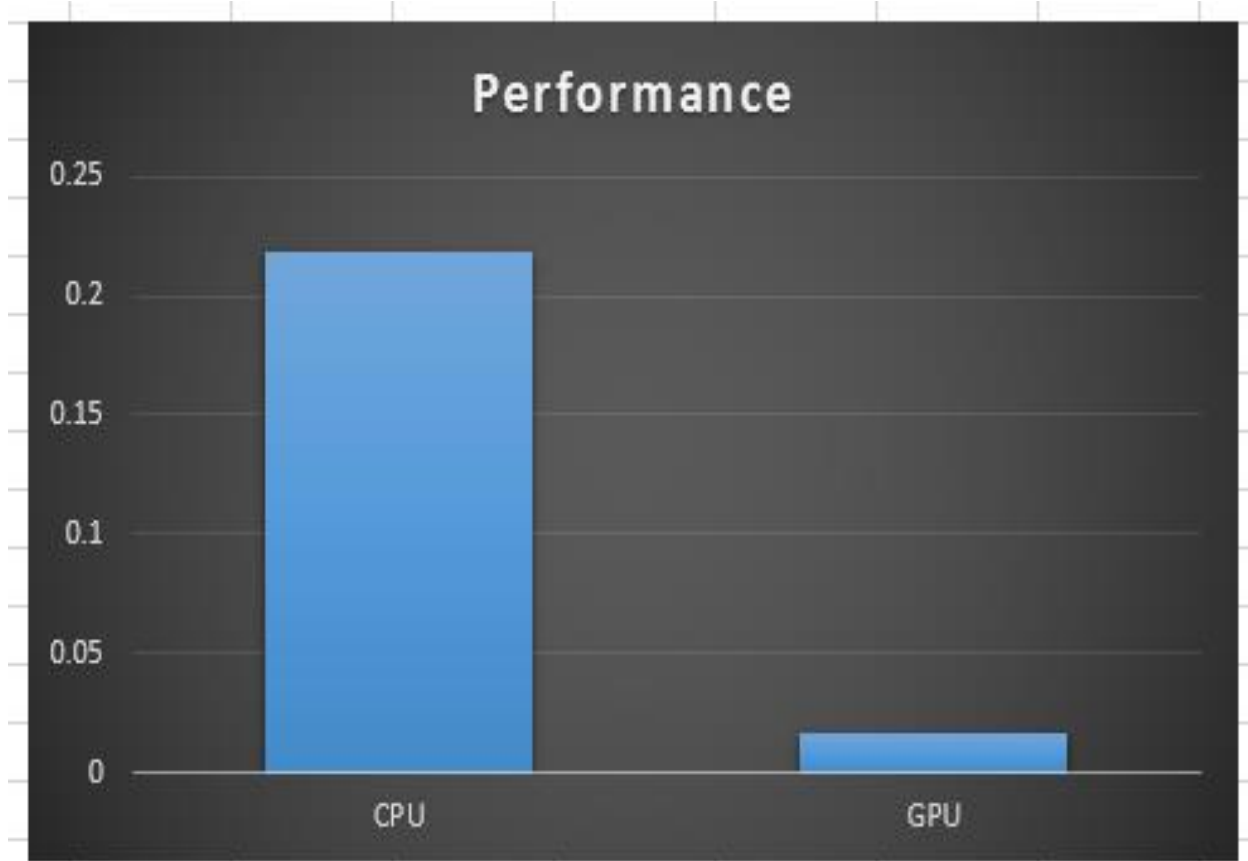


Figure 19: Performance on CPU and GPU

Selection 2: User allocated strings is used as input to be passed into Smith Waterman Algorithm on CPU

```

C:\Users\Anurag\documents\visual studio 2012\Projects\Application1\x64\Deb...
Enter:
  1 - To align two test files
  2 - To align two novel sequences
  3 - To align a novel sequence with a test file
:2
Novel strings should be no longer than 20 symbols. Program is case sensitive
Please enter 1st string:TCAATGAT
Please enter 2nd string:ATTCGACT
  0 A T T C G A C T
0 0 0 0 0 0 0 0 0
T 0 0 5 5 0 0 0 0 5
C 0 0 0 1 10 0 0 5 0
A 0 5 0 0 0 6 5 0 1
A 0 5 1 0 0 0 11 1 0
T 0 0 10 6 0 0 0 7 6
G 0 0 0 6 2 5 0 0 3
A 0 5 0 0 2 0 10 0 0
T 0 0 10 5 0 0 0 6 5
TCAA
TCGA Press any key to continue . . .
Execution time in milliseconds = 0.112 ms

```

Figure 20: Selection 2 results on CPU

Results On GPU

```

C:\Users\Anurag\documents\visual studio 2012\Projects\Application1\x64\Deb...
Enter:
  1 - To align two test files
  2 - To align two novel sequences
  3 - To align a novel sequence with a test file
:2
Novel strings should be no longer than 20 symbols. Program is case sensitive
Please enter 1st string:ATTCGACT
Please enter 2nd string:TCAATGAT
  0 T C A A T G A T
0 0 0 0 0 0 0 0 0
A 0 0 0 5 5 0 0 5 0
T 0 5 0 0 1 10 0 0 10
T 0 5 1 0 0 6 6 0 5
C 0 0 10 0 0 0 2 2 0
G 0 0 0 6 0 0 5 0 0
A 0 0 0 5 11 0 0 10 0
C 0 0 5 0 1 7 0 0 6
T 0 5 0 1 0 6 3 0 5
TCAA
TCGA Press any key to continue . . .
Execution time in milliseconds = 0.011 ms

```

Figure 21: Selection 2 results on GPU

Section 2: Compare CPU and GPU time Result

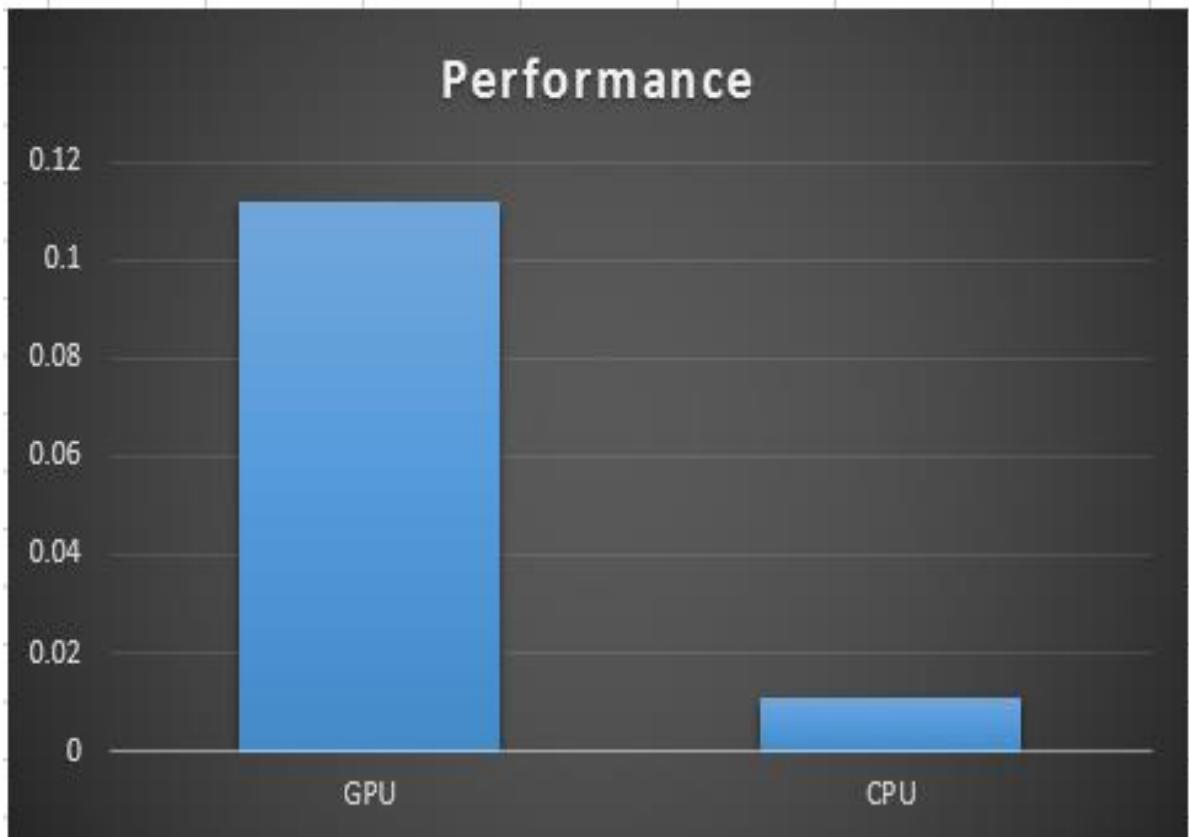


Figure 22: Performance selection 2 on CPU and GPU

Selection 3: User Allocated string as well as file string is used in smith waterman algorithm is used in OpenCL for CPU

```

C:\Users\Anurag\documents\visual studio 2012\Projects\Application1\x64\Deb...
Smith-Waterman Global String Alignment Program

Enter:
  1 - To align two test files
  2 - To align two novel sequences
  3 - To align a novel sequence with a test file
:3
Novel strings should be no longer than 20 symbols. Program is case sensitive
Please enter 1st string:ATTCGACT
  0 A A G C T A G C T A G C
0 0 0 0 0 0 0 0 0 0 0 0 0
A 0 5 5 0 0 0 5 0 0 0 5 0 0
T 0 0 1 1 0 5 0 1 0 5 0 1 0
T 0 0 0 0 0 5 1 0 0 5 1 0 0
C 0 0 0 0 5 0 1 0 5 0 1 0 5
G 0 0 0 5 0 1 0 6 0 1 0 6 0
A 0 5 5 0 1 0 6 0 2 0 6 0 2
C 0 0 1 1 5 0 0 2 5 0 0 2 5
T 0 0 0 0 0 10 0 0 0 10 0 0 1
CT
CT Press any key to continue . . .

Execution time in milliseconds = 0.110 ms

```

Figure 23: Selection 3 on CPU

Selection 3 on GPU

```

C:\Users\Anurag\documents\visual studio 2012\Projects\Application1\x64\Deb...
Smith-Waterman Global String Alignment Program

Enter:
  1 - To align two test files
  2 - To align two novel sequences
  3 - To align a novel sequence with a test file
:3
Novel strings should be no longer than 20 symbols. Program is case sensitive
Please enter 1st string:ATTCGACT
  0 A A G C T A G C T A G C
0 0 0 0 0 0 0 0 0 0 0 0 0
A 0 5 5 0 0 0 5 0 0 0 5 0 0
T 0 0 1 1 0 5 0 1 0 5 0 1 0
T 0 0 0 0 0 5 1 0 0 5 1 0 0
C 0 0 0 0 5 0 1 0 5 0 1 0 5
G 0 0 0 5 0 1 0 6 0 1 0 6 0
A 0 5 5 0 1 0 6 0 2 0 6 0 2
C 0 0 1 1 5 0 0 2 5 0 0 2 5
T 0 0 0 0 0 10 0 0 0 10 0 0 1
CT
CT Press any key to continue . . .

Execution time in milliseconds = 0.012 ms

```

Figure 24: Results of Selection 3 on GPU

Section 3: Compare CPU and GPU time Result

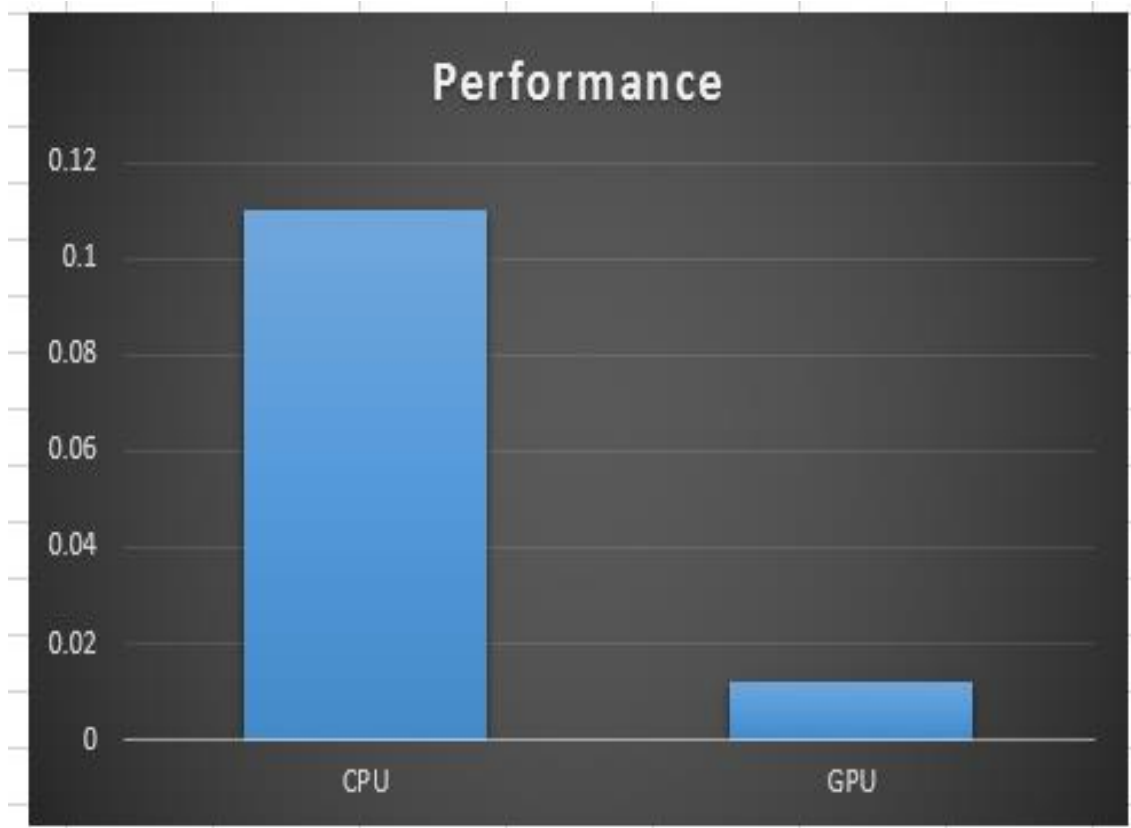


Figure 25: Performance based on Selection 3

OpenCL average time base performance in different devices

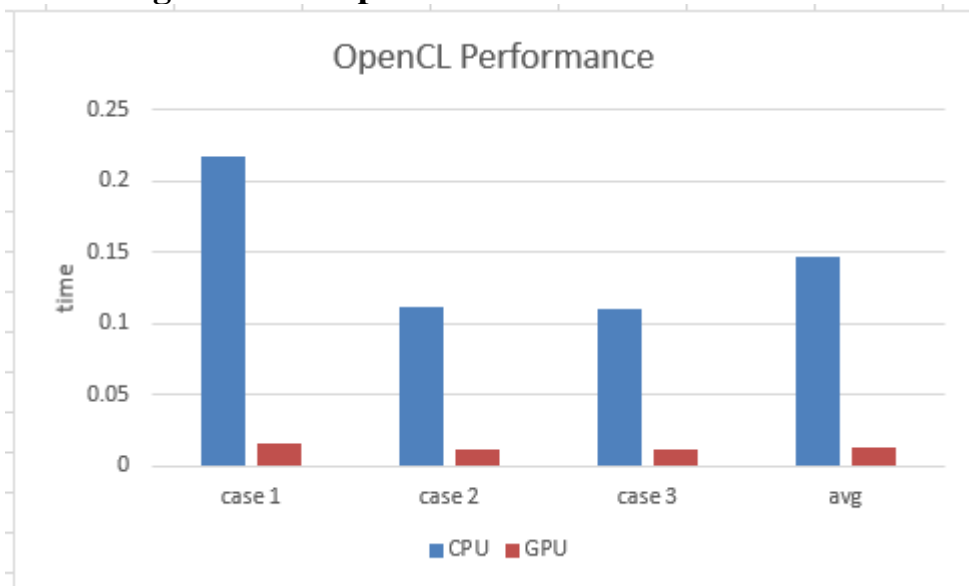


Figure 26: OpenCL avg time performance on devices

CUDA average giga flops based performance in different devices.

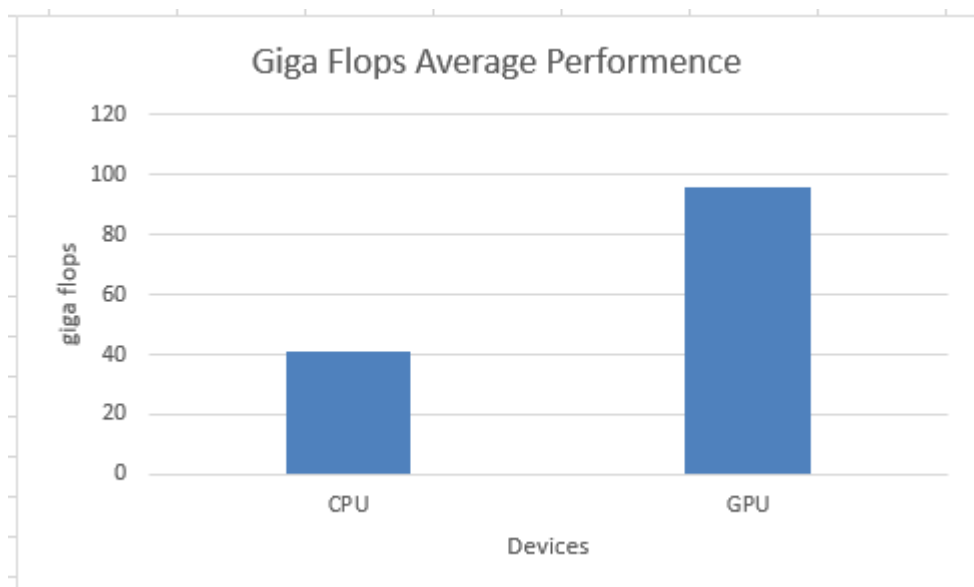


Figure 27: CUDA giga flops average performance on devices

CONCLUSION AND FUTURE SCOPE

The research concludes in proposing OpenCL implementation on visual studio as better technique to solve sequence matching problems. The other technique used in this research is CUDA on MATLAB. The research uses both the techniques to solve DNA sequence matching. On a certain data set of DNA sequence the results from both the implementations is in form of gig flops and time constraint. The gig flops and time constraints results prove OpenCL as better technique. One of the experiments on DNA sequence by OpenCL takes it 0.011 milliseconds to solve it on GPU and 0.112 milliseconds on CPU. The results clearly point out that GPU performs exceptionally better than CPU. Also in one implemented case, at matrix size of 14000 the gip flops taken by OpenCL on GPU is about 60 while the same experimentation on CPU takes about 135 gig flops. Its noted from the experimentation that as the matrix size increases the gig flops for the CPU grows exponentially, thus the research derives the fact that GPU performs exceptionally better than CPU.

FUTURE SCOPE

The future scope of this research remains in implementing the proposed efficient technique, OpenCL and Visual Studio, on many biological tasks. An additional advantage in terms of flexibility that is with OpenCL language and the mechanism it uses in finding the execution path has the potential to get exploited by vast biological applications.

REFERENCES

-
- [1] J. D. Owens, M. Houston, David Luebke,, Simon Green,, John E. Stone, and James C. Phillips, "GPU Computing," *IEEE*, vol. 96, pp. 879-899, 2008.
- [2] r. banger, *OpenCL Programming by Example*, mumbai, 2014.
- [3] "smith waterman in matlab," MATLAB, [Online]. Available: <http://www.google.com/url?q=http%3A%2F%2Fin.mathworks.com%2Fmatlabcentral%2Ffileexchange%2F45831-matlab-audio-analysis-library%2Fcontent%2Flibrary%2FsmithWaterman.m&sa=D&sntz=1&usg=AFQjCNE6SRQ5Bq7PJnKJngvm6Pg7kLfEPw>. [Accessed APR 2015].
- [4] "Measuring GPU Performance," MATLAB, [Online]. Available: <http://www.google.com/url?q=http%3A%2F%2Fin.mathworks.com%2Fhelp%2Fdistcomp%2Fexamples%2Fmeasuring-gpu-performance.html&sa=D&sntz=1&usg=AFQjCNGXOzaRm2k5BYsqpp5RDJTzOyqrBw>. [Accessed APR 2015].
- [5] "cs.stanford.edu," [Online]. Available: http://cs.stanford.edu/people/eroberts/courses/soco/projects/computers-and-the-hgp/smith_waterman.html. [Accessed 20 11 2014].
- [6] "wikipedia," [Online]. Available: http://en.wikipedia.org/wiki/FASTA_format. [Accessed 15 09 2014].
- [7] L. Ligowaski, "an efficient implementation of smith waterman algorithm on GPU using CUDA, for massively scanning of sequence database," *ieeee*, 2009.
- [8] I. hasan, m. kentie and A.-a. zaid , "DOPA: GPU-based protein alignment using database and memory access optimizations," *BioMed Central*, no. 10, 2011.
- [9] Feng Xiaowen, h. jin, r. zheng, l. zhu and w. dai, "ACCELERATING SMITH-WATERMAN ALIGNMENT OF SPECIES-BASED PROTINE SEQUENCE," *springer science+buniess media new york*, no. 10, 2013.
- [10] P. D, vouzis, nikolaos and s. , "GPU-BLAST USING GRAPHICAL PROCESSORS TO ACCELERATE PROTINE SEQUENCE ALIGMENT," *Oxford University Press*, vol. 27, no. 10, pp. 182-188, 2010.
- [11] D. razmyslovich, G. Marcus, M. Gipp, Z. Marc and A. Szillus, "IMPLEMENTATION OF SMITH-WATERMAN ALGORITHM IN OpenCL FOR GPU," *IEEE*, no. 10, pp. 48-56, 2010.
- [12] Edans Flavius de O., Sandes and Alba Cristina M.A. de Melo, "Retrieving Smith-Waterman Alignments with Optimizations for Megabase Biological Sequences Using GPU," *IEEE*, vol. 24, pp. 1009-1021, 2013.
- [13] Yongchao Liu and Bertil Schmidt, "CUSHAW2-GPU:Empowering Faster Gapped Short-Read Alignment Using GPU Computing," *IEEE*, vol. 14, pp. 31-39, 2013.
- [14] Jung-Hyun Hong, Young-Ho Ahn, Byung-Jin Kim and Ki-Seok Chung, "Design of OpenCL Framework for Embedded Multi-core Processors," *IEEE*, vol. 60, no. 2, pp. 233-241, 2014.
- [15] Romain Dolbeau, Francois Bodin and Guillaume Colin de Verdi`ere, "One OpenCL to Rule Them All?," *IEEE*, vol. 13, pp. 1-6, 2013.

- [16] T. Brandes, A. Arnold, T. Soddemann and D. Reith, "CPU vs. GPU - Performance comparison for the Gram-Schmidt algorithm," *Springer*, no. 210, pp. 73-88, 2012.
- [17] Youquan Liu, Shaohui Jiao, Wen Wu and Suvranu De, "GPU Accelerated Fast FEM Deformation Simulation," *IEEE*, pp. 606-609, 2008.
- [18] John D., Mike Houston and David Luebke, "GPU Computing," *IEEE*, vol. 96, no. 5, pp. 879-899, 2011.
- [19] J. Ignacio, Juan Ignacio Perez, E. García and J. A. d. Frutos, "Application of GPU Computing to the Characteristic Basis Function Method," *IEEE*, vol. 919, no. 12, pp. 1003-1006, 2011.
- [20] Changyou Zhang,, Kun Huang, Xiang Cui and Yifeng Chen, "Power-aware Programming with GPU Accelerators," *IEEE*, pp. 2444-2449, 2012.
- [21] Mingcheng Wu,, Jingyi Fu and Xiaorong Hu, "High Throughput TCR Sequence Alignment Using Multi-GPU with Inter-task Parallelization," *IEEE*, vol. 978, no. 12, pp. 231-237, 2012.
- [22] Michael J. Dinneen,, Masoud Khosravani and Andrew Probert, "Using OpenCL for Implementing Simple Parallel Graph Algorithms," *IEEE*, vol. 84, no. 35, pp. 168-174, 2011.
- [23] Ching-Lung Su, Po-Yu Chen, Chun-Chieh Lan, Long-Sheng Huang and Kuo-Hsuan Wu, "Overview and Comparison of OpenCL and CUDA Technology for GPGPU," *IEEE*, pp. 448-452, 2012.
- [24] Jianbin Fang, Ana Lucia Varbanescu and Henk Sips, "A Comprehensive Performance Comparison of CUDA and OpenCL," *IEEE*, pp. 216-226, 2011.
- [25] Baida Zhang, Shuai Xu, Feng Zhang, Yuan Bi and Linqi Huang, "Accelerating MatLab Code using GPU: A Review of Tools and Strategies," *IEEE*, pp. 1875-1879, 2011.

ABBREVIATIONS

- (a) GPU: Graphical Processing Unit.
- (b) CPU: Control Processing Unit.
- (c) AMD: Advance Micro Devices.
- (d) PC: Personal Computer,
- (e) SPMD: Single Program Multiple Data.
- (f) CUDA: Compute Unified Device Architecture.
- (g) OpenCL : Open Computer Language.
- (h) DSP: Digital signal Processing.
- (i) WDDM: Windows Display Driver Model.
- (j) TCC: Tesla Compute Cluster.
- (k) DNA: Deoxyribo Nucleic Acid.
- (l) BLAST: Basic Local Alignment Search Tool.
- (m)GCUPS: graphical cell update per second.
- (n) NCBI: National Center for Biotechnology Information.
- (o) SW ALGO: Smith waterman algorithm.
- (p) NDK: Native Development Kit.
- (q) SDK: Software Development Kit.

Azhar Ali, Balwant Ram, “GPU PERFORMANCE SURVEY ON OPENCL AND CUDA USING SMITH WATERMAN ALGORITHM”, International Journal of Applied Engineering Research (ISSN: 0973-4562).