Accelerated GPU Based Protein Sequence Alignment – An optimized database sequences approach

Muhammad Sadiq Amin[1], Laiq Hassan[1], Awais Aziz Shah[2], Usman Akbar[2], Hafiz Adnan Niaz[3]
[1]Dept of Computer Systems Engineering University of Engineering & Technology Peshawar,Pakistan
[2]Dept. of Computer Science Superior University Lahore,Pakistan
[3]Dept. of Computer Engineering National University of Sciences & Technology, Islamabad ,Pakistan

Abstract
Smith-Waterman (S-W) algorithm is the perfect sequence alignment method for the biological database but practically this algorithm lacks pace due to high computational complexity. FASTA, BLAST and other heuristics approaches are faster in computations but less accurate. Volume and length variation of sequences require restructuring the database. Acceleration of Smith-Waterman algorithm on proper modern hardware brings perfection and accuracy. This paper presents a high-performance sequence alignment algorithm implemented on Kepler's architecture graphic processor unit. This new implementation is improved version having reduced memory accesses to eliminate bandwidth congestion. The implementation is performed on Kepler's architecture graphics processing unit on which the performance was raised to 51 Giga Cells updates per second GCPUS which is 138.3% increase than the previous implementation on GTX275 GPU. In this implementation protein database is converted into equal length sequence sets on advanced GPU. By this workload is distributed among GPU microprocessor threads. This results in improved implementation than previous implementations.

Keywords
Smith Waterman, SwissProt, Proteins, Sequencing, Alignment, GCPUS, FASTA

1. Introduction

Sequence alignment is used to find out the area of likeness in Protein and DNA sequences. Likeness comes in the different sequence is due to evolution, functional, or structural similarities between sequences. Sequence alignment can be global and local [20]. Other approaches are heuristics sequence alignment like FASTA, BLAST, HMMR [15] that are speedy but lack accuracy. Smith-Waterman Algorithm [21] that is based on dynamic programming [22] is however slow for long sequences but results in accurate outcomes for two DNA or protein sequences, for example, a query sequence and database sequence. Normally S-W algorithm is applied on DNA protein strands using CPU clusters but it lacks speed up due to improper distribution of workload [23]. Cell BEs, FPGAs, and GPUs these are the platforms on which S-W algorithm is implemented to get optimal and accelerated solutions [9]. This paper consists of an optimal performance GPU based approach using the database and optimized memory access where we chose Kepler K40 GPU architecture for this purpose. This implementation first converts reference protein into GPU compatible format. Time taking matrix fill step of the Smith-Waterman algorithm is implemented and accelerated on GPU. Optimization and reconstruction of the database on advance GPU do the acceleration job. Memory accesses are also optimized to diminish bandwidth bottleneck. On LMAR having NVIDIA Kepler k40 GPU which contained 2880 cores the performance of GPU accelerated implementation assessed finding were 50 GCPUS (Giga cell updates) for searching October 2015 SWISS-PROT database. This implementation achieved a faster result for protein sequence alignment than the previous implementations.

Before the existence of CUDA, it is the first known SW based implementation using OpenGL to search protein database. There were two implementations one [17] and other was [18]. These approaches were similar to systolic array based on FPGAs [16]. First, the query and database were copied to the device in the form of textures. In an anti-diagonal way, the score matrix is being calculated. A pixel is sketched for every anti-diagonal element. A shader executes for this drawn pixel calculates a score for the cell. The input of next cell is the value that is kept in texture. This technique is similar to systolic array pass values. The implementation [16] shown result of 650 MCPUS for Swiss-Prot database search. Second implementation [18] had two modes one with trace back and another without a trace back.

The result was 241 MCPUS for no trace back and 178 MCPUS with trace back. GPU used for implementation was GeForce 7800GTX. SW-CUDA [19] is first CUDA based SW implantation which is different from systolic array fashion. Each processing element of the device does alignment task. No need for communication between processing elements is an edge that's reducing memory reading and writing tasks. In global memory, the database is stored in such a way that the length is equal, so the threads in a wrap have equal execution time. Query profile

Manuscript received October 5, 2017
Manuscript revised October 20, 2017

is produced where substitution matrix is expanded as the columns of the matrix have query sequence symbols and the rows have protein symbols. A number of alignment score accompanied with query sequence can be perfected when operating on a database symbol. Memory's capability to read and write vectors every kernel operates on four cells.

CUDA (Compute unified device architecture) is a tool used to work with in parallel computing environment using C syntax to launch kernel of GPU. CUDA is designed to support many languages like Fortran, JAVA PYTHON Wrapper, Direct compute, Open ACC. CPU program calls CUDA kernel that is a C like a function with some restriction that invokes device code. Actually, data parallel portion of an algorithm is executed on the device as Kernel. Conventionally only one kernel could be executed at a time but for Kepler architecture this restriction is relaxed. Each kernel is executed by threads. On concept level, CUDA threads are close to data-parallel tasks. CUDA threads are different from CPU threads in the sense these threads are easy to create and extremely light weight.

The designing of CUDA is such a way that it can execute 1000s of threads. These threads form a group that is called Block and these thread blocks are grouped in a grid. Threads block and the grid can be 1D, 2D or 3D in dimensionality depends upon the data structure acts upon. The thread is basic programming unit having its own per thread local memory and has access to registers. Each thread has a unique index in a block and more over each block has a unique index in the following grid. These unique indexes help in the computation of array indices for a particular instance. A thread in a block executes on a single multiprocessor synchronize and share data with the threads in that particular block. The wrap is a combination of threads executed in a multiprocessor which will be a subset of threads from a block. Threads belong to different blocks can be assigned to multiple multiprocessors at the same time or to a particular multiprocessor at the same time that is multithreading or they can be assigned to the same multiprocessor. For Kepler, there can be 32 wraps which are 1024 threads in a thread block.2048 threads or 64 wraps can be active simultaneously on a Kepler multiprocessor. These threads may belong to 2 threads blocks of 32 wraps or 3 thread block of 21 wraps or can be 4 thread blocks of 16 wraps up to 16 blocks of 4 wraps so it means 16 thread blocks can be simultaneously active on a multiprocessor.

The result of this implementation was 1.9 GCPUS compared with software implementation that was 0.12 GCPUS and it was benchmarked on GeForce 8800GTX GPU. Many enhancement and improvements have been suggested to [19] in CUDASW++ [10] 'inter-task parallelization' method are presented for more than 3,075 amino acids that need little memory and slower. This approach is benchmarked on GeForce 280GTX device achieves 9.5 GCPUS speed while searching Swiss-Prot. An advanced version CUDASW++2.0 [9] has been coming forth to publish. This approach achieved 17 GCPUS that was benchmarked on GTX280. A recent approach is DOPA [14] which was benchmarked on GTX275 GPU achieved 21 GCPUS. In this implementation, data is kept in registers and the conversion of the database in an interlaced manner to fit in GPU. Accessing global memory is slow so avoid access to global memory is avoided for unnecessary usage. The improvement was measured 138.3% increase in GCPUS when implemented on aforementioned GPU. The execution time was 0.0086 seconds in average for a query sequence that is very less than previously implemented DOPA.

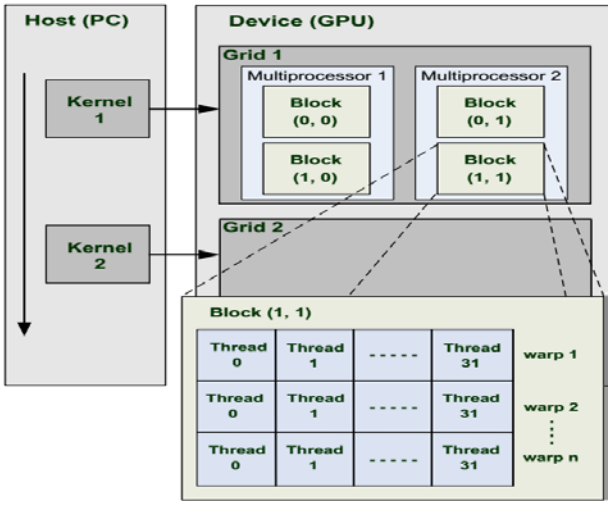

Fig.1. Programming model. CUDA hierarchy of threads, blocks.

2. METHODOLOGY

The method used in the implementation for high performance and optimized TelsaK40 GPU based protein sequence alignment are presented as follows. NAVDIA CUDA a mature GPU programing language toolkit for device coding with conjugation of C++ for host PC having GENTO installed. Most of the previous implementations were done using CUDA. Protein searches will be in main focus despite DNA and Swiss-Prot database will be used. Searching for proteins in the database is complex due to substitution matrix. In this implementation, we are interested to return maximum Smith-Waterman score rather than actual alignment. So reducing the trace back step in SW greatly simplifies the implementation. Pointers

are stored to reduce memory consumption. Additionally to generate complete alignment those sequence having highest score a new database will be filled with. The new database will be searched by some CPU based FASTA suit SS search program.

This technique may lead to some redundancy because some sequence may align twice but the count is shallow. Swiss-Prot contains 550,000 sequences while we will return top 20 sequences. There are various approaches to achieve sequence parallelism like systolic array manner in which data is kept passing between processing elements to search particular sequence. Secondly, the approach is processing element do alignment task in a parallel manner by multiple elements. Both approaches can be mingled where many processing elements are combinable to perform alignment task for large sequences. Another approach that is each processing element performs complete alignment task was considered. This approach greatly helps in getting rid of communication among processing elements. It simplifies the implementation and best way to utilize resources. GPU used in our implementation contains 2880 cores While Swiss-Prot contains 550,000 sequences up till this writing so it is the best way to keep all processing element well occupied by this approach.

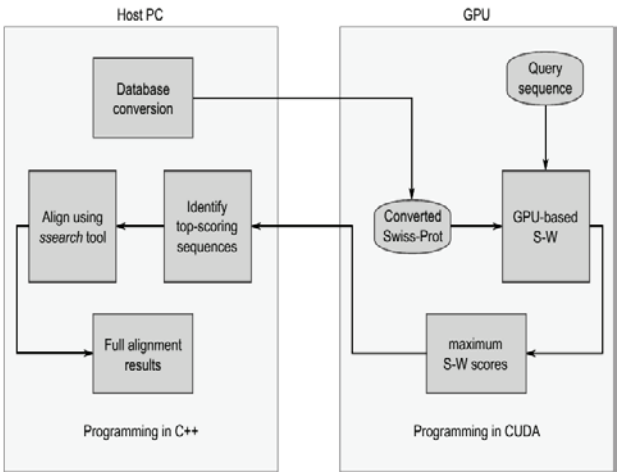

Fig.2. The figure presents a block diagram description of the GPU implementation.

a. Database Conversion

FASTA format, sequence consists of sequence description and the biological information about them. The database is priory converted by GPU implementation to a GPU compatible format to reflect good match for GPU capabilities. The need for conversion of the database is just once it is then stored in new format locally. The database conversion time is less than a second. The conversion process is shown in figure 5. Contains steps below.

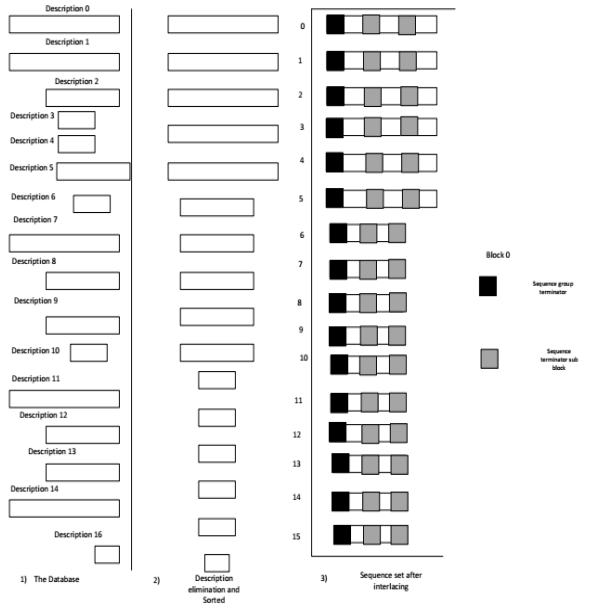

Fig.3. The figure presents represents database conversion process

b. Description separation and sorting

To reduce the wastage of processing time by waiting of half-wraps threads by waiting for each other for completion of processing time instead of carrying on independently database sequences are divided by lengths to reduce the difference in lengths between adjacent threads. Another file is made for sequence description to be kept there; the description is not loaded to device resulting conserved memory ultimately reduced loading time.

c. Sequence grouping

After description separation and sorting by lengths distributed equal workload in every sequence set but some sequence set remained unequal in lengths. To avoid this inequality sequences in sequence set are linked up with relic sequence to make sequence groups in such a way that the total length in sequence group in a sequence set is almost same as the lengthy sequence in the set. Tasks under process are equalized for half wrap's threads for sequence set. To assure device Kernel to start next alignment sequence terminators are introduced among linked up sequences. Linked up sequences ending points are indicated by terminators of sequence groups. At this stage, the thread within half wrap will wait for each other to halt the execution.

d. Interlaced sub grouping

A new file is written in an interlaced manner that contains 16 sets of sequence groups from database sequences. Every subgroup contains 8 characters from every sequence in a way that 8 characters are taken from set's group first then 8 characters from next group and so forth. Each sequence set contains sixteen sequence groups in a particular sequence set and every half-wrap thread is made able to load eight-byte sequence data from adjacent addresses achieving coalesced loading of 128-bytes

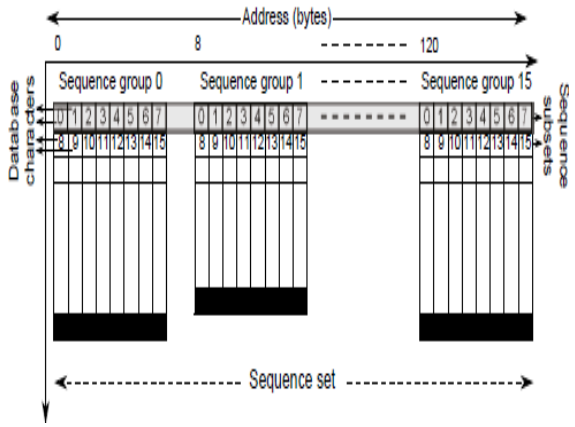

Fig.4. The figure shows sequence storing as interlaced subsets.

e. Temporary read and writes

During GPU implementation Memory bandwidth that is represented was a critical logjam. To accelerate and optimize to achieve enhanced performance many steps were taken. S-W matrix values are of no need to be kept and can be overwritten when there is no trace back step is taken in place on GPU.

$$Hi,j = \max\begin{cases} 0 \\ Hi-1,j-1+Si,j \\ Di,j \\ Ei,j \end{cases} \quad (1)$$

Where

$$Di,j = \max\begin{cases} Hi-1,j-\propto \\ Di-1,j-\beta \end{cases} \quad (2)$$

$$Ei,j = max\begin{cases} Hi-1,j-\propto \\ Ei,j-1-\beta \end{cases} \quad (3)$$

A single column that keeps value to left of column f (i, j-1) that is under operation presently. For reduced memory usage column size is set to a sequence of the query rather than DB sequence. Upon starting of every new database sequence this column can be reset to zero. Other variables are used to store values of top, top left cells as stated in the algorithm. The query sequence is approached in the interior loop rather than database sequence. It is good for keeping in the fast memory innermost sequence is approached for each individual outermost sequence's alphabet.

A single query sequence is best applicant for this instead of a database having bulk number sequences. A new temporary column is introduced to store Ix values and upper Iy value is stored in the case of affine gap penalties. To read and write score and Ix temporary values four accesses took place in each S-W iteration. For every access 32 bytes read/write is granted when both are in non-coalesced form. That meant 2048 bytes per half-wrap bandwidth was in use.

2 read/write*2 value * 32 byte*16 thread/ half-wrap = 2048 bytes.

That result in a serious memory logjam. To squeeze this to one 128 reads/write for each 2nd repetition some moves were made. First, a 16-bit unsigned short data type was used for temporary values that theoretically decrease needed bandwidth to half and allows good coalescing afterward. Coalescing was a 2nd move which was making every thread to keep single temp score. A pointer was started into temporary storage despite direct array accesses at the Thread id and increased by all threads to pass over to next cell. Every thread per half-wrap reads a 16-bit coalesced value now despite two 256-bit accesses two accesses happen in a half-wrap which improves 16x bandwidth and caused 10x acceleration. Thirdly by definition, a Temp data structure that contains 2 unsigned short data types to access score and Ix in a single iteration half the memory accesses. At this stage, half-wrap would access 2 values in a read for 64-byte bandwidth that causes 512-bit coalesced accesses. Finally, 2 temporary values are interleaved to go to 1024-bit accesses.

These steps improved 16xbandwidth and need 1 access instead of sixty-four. Maximum allowed coalesced access size is 1024-bits that is efficient than many smaller accesses in coalesced manner.

f. Substitution matrix accesses

In the case of protein alignment, substitution matrix is critical due to its access time because of the alignment of two symbols every time. It accesses are random for example BLOSUM 62 making the selection of memory usage complicated. Substitution matrix is based on database sequence. Global memory has much access time making it a worse candidate to use. Coalescing is also become misery due to the randomness of substitution matrix accesses. Texture memory is the right candidate to store substitution matrix due to its low latency .

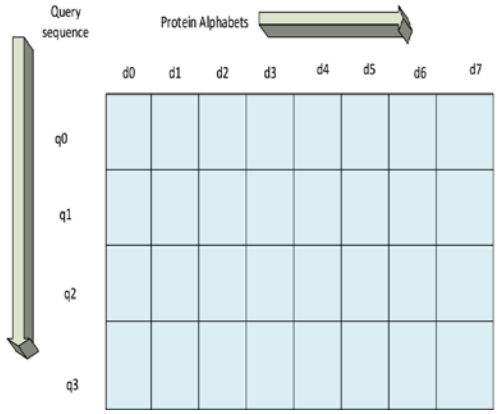

Fig.5. the figure describes memory accesses to fetch values from a query profile

Texture memory can be used in such a way that it can fetch 4 values of substitution matrix from query profile. To eliminate the random nature of substitution matrix for a given database the query sequence used on the top row despite protein symbols. For query alignment with a database character, many substitution scores can be loaded. Query profile is produced one time for each query sequence so query sequence retrieval is eliminated the only present position in a query is required for profile indexing. Every query profile keeps value for twenty-three characters. So the columns and memory needed for query profile are dependent on query sequence length. Kepler 110b used for this implementation has 48KB texture cache.

3. Results

Increase cache miss rate for query sequence more than will be [48 * 1024/23] = 20137 characters. Performance elevated about 25% on Kepler k40 architecture by using query profile with Swiss-Prot.

The performance of protein sequence alignment based on Smith-Waterman implemented on advanced GPU (Kepler K40) is evaluated and the result is compared with previous implemented approaches are discussed below.

It was decided to perform implementation on LMAR. The system had Gentoo (Linux) operating system installed. The GPU on board was NVIDIA Kepler K40 device that has s 2880 cores while having 12 GB GDDR5 memory with a clock speed of 875MH. Programming for the device is done using CUDA toolkit version 6. While for host programming is done using visual studio with C++ as language chosen. The database used was Swiss-Prot October 2015. Substitution matrix was set to BLOSUM64 and GAP penalty was set as -10 while Gap extended penalty was set to -2 though this doesn't affect program execution. C clock () instruction was used to determine run time which was later verified by CUDA profiling application. The measured time doesn't include database loading time the reason is the size of the database. Often loading time of data set exceeds due to its bulk size as compared to alignment time. The database is loaded only one time eliminating the overhead making easy to search query sequences. The measured results were benchmarked in GCPUS which was about average of 51 in 8 milliseconds. Swiss-Prot performance comparisons with different previous implementation on various GPU are described in this section. With NVIDIA Kepler K40 GPU 50 GCPU were benchmarked in our implementation.

- Comparison with BLAST based on heuristic method was benchmarked on Q6600 processor manufactured by Intel obtain 15.3 GCPUS [17].
- Comparison with CUDASW++ has first known GPU implementation based on smith-Waterman implemented using GeForce GTX 280 GPU and GeForce GTX 295 achieved 9.66 and 16.08 GCPUS [10].
- Comparison with CUDASW++2.0 was implemented on GeForce GTX 280 GPU released in 2010. It was improved by 1.74 times then CUDASW++ by using SIMT algorithm described in [9].
- Comparison with DOPA was implemented on GeForce GTX 275.It was 1.75 times faster than CUDASW++2.0 and achieved 21.4 GCPUS [14].

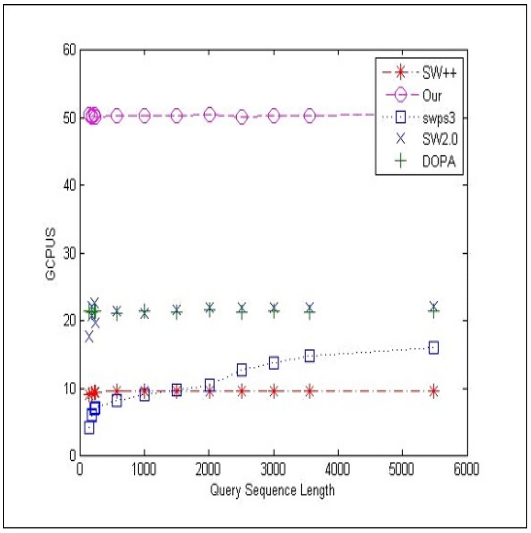

Fig.6 graph shows comparison Previous mentioned implementation vs ours

The performance of our implementation on NVIDIA Kepler K40 GPU achieved higher GCPUS and attend less execution time as compare to all aforementioned

implementations that 51 GCPUS. Furthermore, our implementation due to hardware advancement is faster than previous implementations.

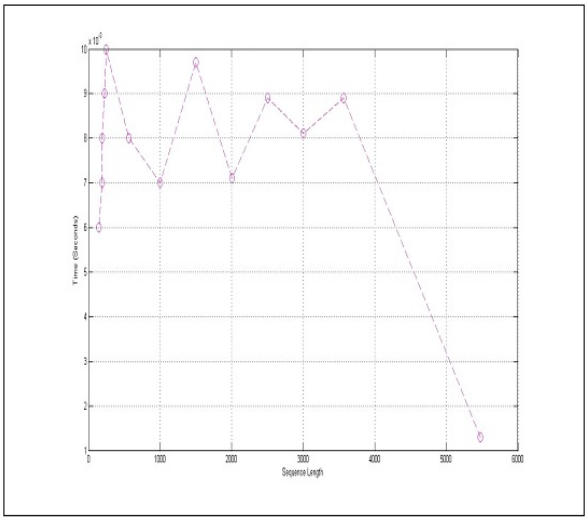

Fig.7. Graph shows query sequence length and execution time

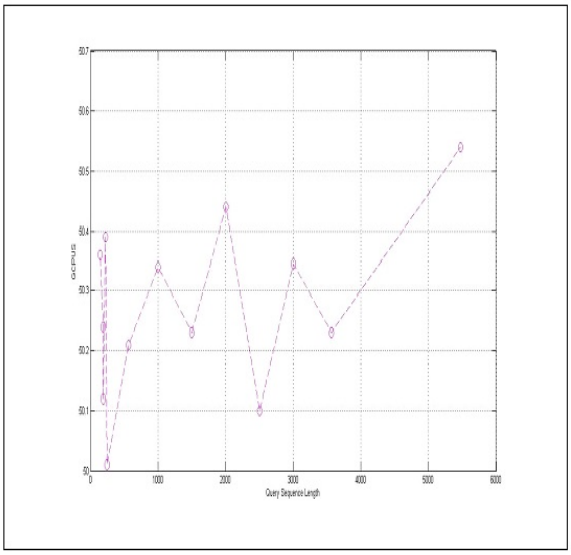

Fig.8. the graph shows query sequence length and performance in GCPUS.

Table 1: Performance results with Swiss-Prot

Sequence	Length	Execution time (Sec)	GCPUS
P02232	144	0.0067	50.36
P05013	189	0.0075	50.24
P14942	222	0.0084	50.12
P20930	4061	0.0093	50.39
Q9UKN1	5478	0.012	50.01

4. conclusion

The improvement was measured 138.3% increase in GCPUS when implemented on aforementioned GPU. The execution time was 0.0086 seconds in average for a query sequence that is very less than previously implemented DOPA. Furthermore, the performance of our implementation on aforementioned GPU resulted in 428% increase in GCPUS as compared to CUDASW++ and 200% increased than CUDASW++2.0.Besides performance, it is good to buy a GPU for high computing and better task processing when one comes across to pay for CPU. On the other side, the power consumption of a GPU is higher than CPU which makes the performance/Watt nearly equal. CPU is very flexible rather than GPU but when Parallel (device accelerated) tasks are about to perform an ordinary CPU with a GPU is enough. Another thing is GPU based solution development is a cumbersome job than development for a CPU. Additional communication over the bus to GPU and low memory makes it less flexible.

Lastly, in conclusion, GPU based solutions are time-consuming during development stage have more power consumption and additional price expenditure for GPU than CPU. However if one offers a better protein searching performance than CPU while don't want to stick with dedicated FPGA based solution GPU is a better option in hand.

References

[1] Zhang, J., Misra, S., Wang, H. and Feng, W. (2016). M blast: database-indexed protein sequence search on multicore CPUs. BMC Bioinformatics, 17(1).

[2] Shah, H., Hasan, L. and Koo, I. (2016). Optimized and Portable FPGA-Based Systolic Cell Architecture for Smith-Waterman-Based DNA Sequence Alignment. Journal of information and communication convergence engineering, 14(1), pp.26-34.
[3] Okada, D., Ino, F. and Hagihara, K. (2015). Accelerating the Smith-Waterman algorithm with interpair pruning and band optimization for the all-pairs comparison of base sequences. BMC Bioinformatics, 16(1).
[4] Liu, Y., Hong, Y., Lin, C. and Hung, C. (2015). Accelerating Smith-Waterman Alignment for Protein Database Search Using Frequency Distance Filtration Scheme Based on CPU-GPU Collaborative System. International Journal of Genomics, 2015, pp.1-12.
[5] Lan, H., Chan, Y., Xu, K., Schmidt, B., Peng, S. and Liu, W. (2016). Parallel algorithms for large-scale biological sequence alignment on Xeon-Phi based clusters. BMC Bioinformatics, 17(S9).R. Nicole, “Title of paper with only first word capitalized,” J. Name Stand. Abbrev., in the press.
[6] Jiang, H. and Ganesan, N. (2016). CUDAMPF: a multi-tiered parallel framework for accelerating protein sequence search in HMMER on CUDA-enabled GPU. BMC Bioinformatics, 17(1).
[7] Huang, L., Wu, C., Lai, L. and Li, Y. (2015). Improving the Mapping of Smith-Waterman Sequence Database Searches onto CUDA-Enabled GPUs. BioMed Research International, 2015, pp.1-10.
[8]
[9] Chen, X., Wang, C., Tang, S., Yu, C. and Zou, Q. (2017). CMSA: a heterogeneous CPU/GPU computing system for multiple similar RNA/DNA sequence alignment. BMC Bioinformatics, 18(1).
[10] Y. Liu, B. Schmidt and D. Maskell, "CUDASW++2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions", BMC Research Notes, vol. 3, no. 1, p. 93, 2010.
[11] Y. Liu, D. Maskell and B. Schmidt, "CUDASW++: optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units", BMC Research Notes, vol. 2, no. 1, p. 73, 2009.
[12] A. Akoglu and G. Striemer, "Scalable and highly parallel implementation of Smith-Waterman on graphics processing unit using CUDA", Cluster Computing, vol. 12, no. 3, pp. 341-352, 2009.
[13] Liu Weiguo, B. Schmidt, G. Voss and W. Muller-Wittig, "Streaming Algorithms for Biological Sequence Alignment on GPUs", IEEE Trans. Parallel Distrib. Syst., vol. 18, no. 9, pp. 1270-1281, 2007.
[14] T. Oliver, B. Schmidt, and D. Maskell, "Reconfigurable architectures for bio-sequence database scanning on FPGAs", IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 52, no. 12, pp. 851-855, 2005.
[15] L. Hasan, M. Kentie, and Z. Al-Ars, "GPU-accelerated protein sequence alignment", 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 2011.
[16] Finn, Robert D., Jody Clements, and Sean R. Eddy. "HMMER web server: interactive sequence similarity searching." Nucleic acids research 39.suppl 2 (2011): W29-W37.
[17] Chow, E., Peterson, J., Waterman, M., Hunkapiller, T. and Zimmermann, B. (1991). A systolic array processor for biological information signal processing. Proceedings of the 5th international conference on Supercomputing - ICS '91.
[18] Weiguo Liu, Schmidt, B., Voss, G., Schroder, A. and Muller-Wittig, W. (2006). Bio-sequence database scanning on a GPU. Proceedings 20th IEEE International Parallel & Distributed Processing Symposium.
[19] Liu, Y., Huang, W., Johnson, J. and Vaidya, S. (2006). GPU Accelerated Smith-Waterman. Computational Science – ICCS 2006, pp.188-195.
[20] Manavski, S. and Valle, G. (2008). CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC Bioinformatics, 9(Suppl 2), p.S10.
[21] Hasan, L., Al-Ars, Z. and Vassiliadis, S. (2007). Hardware acceleration of sequence alignment algorithms-an overview. 2007 International Conference on Design & Technology of Integrated Systems in Nanoscale Era.
[22] Smith, T.F., and Waterman, M.S. (1981) Identification of Common Molecular Subsequences. Journal of Molecular Biology, 147, 195-197
[23] Giegerich, R. (2000). A systematic approach to dynamic programming in bioinformatics. Bioinformatics, 16(8), pp.665-677.
[24] Ebedes, J. and Datta, A. (2004). Multiple sequence alignment in parallel on a workstation cluster. Bioinformatics, 20(7), pp.1193-1195.