Indian Journal of Science and Technology, *Vol 9(45), DOI: 10.17485/ijst/2016/v9i45/106351, December 2016*

ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Protein-protein Interaction Prediction using Variable Length Patterns with GPU

Jeevan Bala, Avinash Kaur* and Parminder Singh

Department of computer Science Engineering, Lovely Professional University, Phagwara - 144411, Punjab, India; jeevan94.47@gmail.com, avinash.14557@lpu.co.in, parminder.16479@lpu.co.in

Abstract

Objectives: To design a new framework to efficiently parallelize the steps of VLASPD algorithm using a hybridized apriori and fp-growth on GPU; to implement the existing and proposed framework in CUDA;to improve the performance factors like computational time, memory and CPU utilization.**Methods/Statistical Analysis**: This paper proposes the acceleration of Protein-Protein Interactions (PPIs) prediction on Graphics Processing Units(GPUs). A GPU can provide more processing cores and computational power in the same cost as a CPU.**Findings**: The frequently occurring patterns in the protein sequences can be used for PPIs prediction.The moving of the approaches from fixed length to variable length lead to computational complexity but also is found to be advantageous.**Applications/Improvements:**Sequence biology is since being researched by various computer engineers, the GPUs can be employed for predicting various sequence interactions like DNA-Proteins, etc. Since the GPU runs the parallel code efficiently, the methodology can be further improved if efficiently parallelized.

Keywords: Parallel VLASPD, Protein Interaction Prediction, Protein Sequences, Variable Length Patterns

1. Introduction

Bioinformatics is the study and understanding of biological data with the help of computer science, mathematics, engineering and statistics. This biological data could be protein sequences, genomes, nucleotides and many more. Bioinformatics lets us see life at molecular level with a clear vision[1].A Protein sequence is a chain combination of amino acids connected by peptide bonds. These amino acids come together in chains to form a protein. Different combinations of amino acids would form different proteins but only a handful (~500) of valid combinations is there which form proteins[2]. These sequences are too long and complex to do remember or determine. So, the amino acid data is fed to a computer system which further gives out a protein as the output. Large computer databases of the sequences are kept[3].The approach of biological data processing has now moved to computational biology, the manual work has been reduced to a large extent and computational researchers has taken place of the scientists those performing experimental approaches[4].

These sequences are important to the structural representations since they can be used to predict the behavior of proteins, like interaction among the proteins with less complexity[5]. The sequences are searched for similarities in them, or co-occurrence of patterns. Various algorithms have been developed by the researchers, which uses the concept of 'k-mers' to predict the interaction among the proteins. 'k-mers' are fixed length patterns found in sequences. Forming associations between the frequently occurring patterns are being efficiently used to predict the interaction among protein sequences. The recent algorithm called VLASPD (Variable length Associative Sequential Pattern Discovery) emphasizes on searching for frequently occurring 'variable length patterns' in the database, and then finding the associations among them[5]. The significant associations then predict the interactions among the proteins. Although VLASPD provides a new research but this can be further improved by the efficient use of data structures.

*Author for correspondence

As the technology is advancing and new data is being researched, there is a need of a fast and reliable computer system which can do the complex calculations as fast as possible with as little effort as possible. To achieve this, a GPU (Graphics processing unit) would be an efficient choice to base the computer system. A GPU is a specialized processor used to accelerate the processing of images and graphics. A GPU can provide more processing cores and computational power in the same cost as a CPU. This is because they have a high parallel structure which makes them very efficient for processing algorithms and data in parallel[6]. The paper also proposes a novel parallel algorithm for frequent patterns mining and forming associations on GPU.

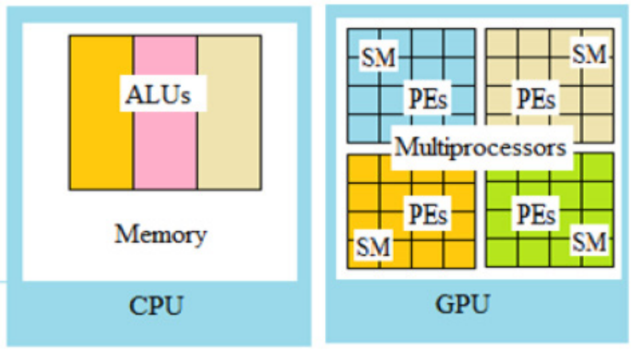

Figure 1. **(a)** The diagrammatic view of CPU containing multiple parallel processing cores, whereas **(b)** GPUs contains hundreds of cores.

1.1 GPUs Accelerate Applications

A fairly recent advancement in the field of bioinformatics is the appliance of Graphical Processing Units (GPU) that uses thousands of cores, and thus increases computing power as compared to the CPU. Initially planned for video gaming & graphics, GPUs have turned out to be capable and adaptable for complex computing applications. These can be used for general purpose computing and simulations such as fluid dynamics, rigid-body physics, and many more[3]. Modern GPUs are exceptionally parallelized floating point stream processors(NVIDIA latest graphic cards can produce 20 GFLOPS for each watt) which can provide un-pretended performance by running compute-intensive tasks on GPU, while the rest of the code still keeps running on the CPU[6]. GPUs comprise of processing elements that are more efficient than those of parallel CPUs. These Processing Elements(PEs) are assembled into various multiprocessors that provide facilities like shared memory[6]. Figure 1 shows the basic structure of modern GPU. The number of processing elements varies in various multiprocessors. Multiprocessor's elements perform same operation on the data: thus called Single Instruction Multiple Thread(SIMT) processors.

The multiple cores embedded in a GPU and SIMD processing means that GPU has a large number of ALUs in comparison to CPU. SIMD width(number of working items processed by GPU thread) over GPU is thus more.

1.2 Compute Unified Device Architecture(CUDA)

NVIDIA released CUDA in 2007, which is the platform and the API (Application Programming Interface) that allows the programmers to use the commonly used languages like C, C++ to access NVIDIA GPU's resources. CUDA architecture, as shown in Figure 2 contains several blocks, each containing a number of threads. The number of threads in a block can be chosen by the programmer[6]. The block level threads, executing parallel can be called as Warp. The blocks are grouped into grids. All the blocks in a grid are of same shape and size. The execution of threads in a same block is done by same microprocessor, which can also process other blocks simultaneously.

The proposed methodology aims to design a new framework to efficiently parallelize the steps of VLASPD algorithm with GPU.

A method is proposedthat used k-mers to predict the protein-protein interactions. The method first traversed the entire protein database to find 3-mers, i.e. all the possible patterns of amino acids of size three. A pairwise kernel function checked the similarities between every pair in the database. Then, the Support Vector Machine(SVM) was used to distinguish between the proteins that could interact, and others that could not[7].

Another approach isdeveloped for PPI prediction, which is the first method for successfully predicting protein interactions by using only sequence information. This approach uses machine learning techniques for predictions. The method used s-kernel instead of pairwise kernel that used s-kernel function instead of pairwise function. The method differed from that of Beh-Hur's method was the kernel used. This method made use of SVM[8]. Some authors[5] recently claimed that the patterns of variable length in the protein sequences can be efficiently used to predict the interactions between the protein sequences. The proposed method involved three

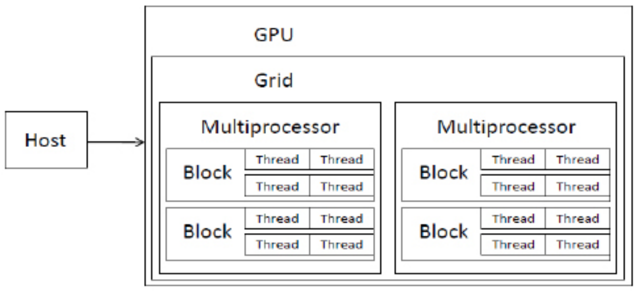

Figure 2. CUDA hierarchy.

steps. The first involved the identification of frequently occurring patterns of 'variable sizes'. The first step used apriori algorithm. The second step finds out all the possible associations between all the possible pairs of frequently identified frequent patterns. The third step involved the identification of the significant associations. The final step after this step predicted the interactions between the protein sequences in the database.

The literature review included some papers on acceleration of sequence alignments, those using Smith Waterman algorithm on GPU. These are included in this section.

SW algorithm[9] is implemented an SW algorithm on graphics card for the first time by using graphic API (OpenGL + GLSL) to gain high performance. The alignment method ran 9 to 15 times faster than other database searches like OSEARCH[3] and SSEARCH[3]. The approach used was: First copy the query and database sequences as textures to GPU memory, then operate the score matrix in an anti-diagonal manner and a pixel is drawn, which executes a pixel shader, which calculates score for the cell. The results act as input in next pass. The method offered execution in two modes, i.e. with and without trace back. The version with no trace back showed performance as 241 MCUPS, 178 with trace back, 120 GCUPS was the performance on CPU.

The first usage of CUDA[10] is implemented for aligning protein sequences, 'SW-CUDA'. The complete alignment task was performed by each of the GPU processing elements, rather than processing as a single matrix. Major advantage gained here was the reduced memory accesses, since no communication between the processing elements was required. The method generated a query profile, which was a query specific substitution matrix (query elements taken as columns),generated once for the entire database. The access to query profile was much less than that of substitution matrix. SW-CUDA achieved 1.8 GCUPS on single and 3.48GCUPs on dual GeForce 8800 GT GPU while searching Swiss-Prot, which was 2 to 30 times faster than the previous implementations on GPU taken for comparison. However, the usage of Manavski and Valle, while conveying great performance, could not completely exploit the capability of the hardware.

Manavski algorithm isimproved[11]. The project claimed many drawbacks in the previous algorithms, which are illustrated: Using texture memory of GPU to store query profile fills it up quickly, thus caused cache misses and memory delays as a result. Query length larger than 356 caused cache misses.The necessary multiple launches of kernel due to large number of database sequences to process limited grid size also caused latency.

Akoglu algorithm[11] used two cached memories on GPU, texture cache and constant cache. It eliminated the use of CPU completely. A function using ASCII code table accessed the score in substitution matrix. Storing the substitution matrix and query sequence in constant memory since access to constant memory, made the algorithm execute faster, the similarity scores were stored in shard memory, which even made the algorithm faster. The algorithm could thus reduce bottlenecks. The algorithm used 64 threads per block as in case of Manavski algorithm. SW score in this case was calculated as four cells at a time. Global memory stored cell calculations simultaneously as updated.

CUDASW++ versions for protein database search onto NVIDIA GPUs is proposed[12–14]. These proved to be one of the key undertakings for implementation of the Smith-Waterman algorithm on GPUs, the source code of CUDASW++ is openly accessible.

CUDASW++ 1.0[12] performed all the Smith-Waterman calculations on GPUs by using the multiple G200 (and higher) GPUs for performance enhancement. The sequences of length less than threshold length were aligned using intertask kernel (uses single thread to align). And, those sequences of length greater than threshold were aligned using intratask kernel (used thread block to compute the alignment). The intratask method imposed communication between the threads. Intertask kernel could achieve better performance due to high parallelism. The average performance of intertask kernel was approximately 17 GigaCUPS(GCUPs) while that of intratask kernel was 1.5 GCUPs when same query was compared on same database sequence on the Tesla C1060. This performance dropped with increase in threshold.

Work is published[13] on protein database search on GPU at the same time as that of CUDASW++, they proposed how to utilize shared memory to enhance the performance of the Smith-Waterman algorithm. The project did not implement intra-task parallelism for the long sequences. The results were compared to parallel implementation of cell processor[14], which provided highest performance of Smith-Waterman approximately 9 GCUPs on single CPU. The measured performance of Ligowski algorithm on NVIDIA 9800 GX2 was 14.5 GCUPS, which was thus measured as the leading performance among those algorithms of those times. The paper claimed its performance factor roughly be double of Farrar implementation[15] and was 50 percent more than the previous version of same algorithm on Sony PlayStation3[14].

Parallel scan algorithm is porposed[16] and claimed that the classical diagonal parallelism undergoes non-uniform parallelism distribution and the memory access is difficult. Thus, memory coalescing is less advantageous in that case. They aimed to fully parallelize the processing of cells in a row of matrix, for which, parallel scan was performed to update the cells. In order to impose the data dependence between the cells in the same row, they performed the parallel scan for updating the values of the cells, which required high cost for synchronization between threads and thread blocks. The parallel scan algorithm could also be employed in intratask kernel of CUDASW++.

An enhanced version of CUDASW++ is produced[17], and named it as CUDASW++ 2.0, 2010, enhanced the performance of CUDASW++ 1.0 based on SIMT abstraction. The performance factor was increased to 17 GCUPs on single NVIDIA GeForce 280 and 30 on dual NVIDIA GeForce GTX 295.

The improvement in backtracking procedure is performed[18]. The method used four Boolean matrices to store the directions of backward moves during backtracking process. The method can be applied to CUDASW++, to improve the performance. The results were not as good as CUDASW2.0.

Intrataskparallelizationis used[16] and multiple GPUs to propose a CUDA based SW algorithm, CUDA-SSCA\#1.

Another algorithm (HKA algorithm) is proposed[19] by using new sequence database organization and several optimizations on GPU to reduce memory accesses. Method pre-converted the sequences into the format that was easy to access. The sequences were first sorted, and then sequence sets and sequence groups were formed, which were formed by concatenating the sequences. The data read writes were reduced to reduce the memory access, thus increasing the performance. The algorithm was 1.13 times better than CUDASW++2.0.

Another attempt is made to improve CUDA++. The authors[20] proposed a tiling approach for performance enhancement of the intratask kernel of CUDASW++. The paper called attention to a few essential configuration issues for tuning execution, including how to ensure that registers are used, rather than global memory, even with exceeded capacity. The improved intra-task kernel when employed into CUDASW++ increased the average performance by 4 GCUPs or 25%.

CUDASW++ 3.0 used CUDA C++[21] and PTX low level computing constructs, which focused on GPUs build around Kepler building design. It performed simultaneous CPU and GPU calculations to speed up the SW algorithm, by dynamically distributing the workload among them to balance the execution times. The method implemented Multithreading and SSE based instructions are implemented on CPU side and PTX SIMD instructions were implemented on GPU to parallelize the Smith-Waterman algorithm.

Another method isproposedcalled CUDA-SWf (an ITE technique)[22] to compare a query sequence to multiple database sequences, that used Frequency Distant Filtration Scheme(FDFS) to filter unnecessary sequences at run time. FDFS used GPU to first calculate the frequency vectors for the query and for the database sequences. Secondly it calculates the frequency distance between the queries and database sequences. Thirdly the sequences that are needed to be compared are transferred from to GPU to CPU. Finally, sorting of sequences takes place in CPU, and the sorted sequences are retransmitted to GPU for SW computations.

An enhanced version of CUDA-SWf as CUDA-SWfr is introduced[23]. The paper performed protein database search on CPU-GPU collaborative system that used Intratask parallelization technique and improvedthe statistical measure and efficiency of the previous algorithm.

A method isintroducedfor efficiently mapping short query sequences. The[24]algorithm focused on performance enhancement by making the database organization more effective; the GPU threads were increased for every database sequence; and minimizing the memory access to avoid bottlenecks.

2. Scope of the Study

Proteins as complex structures are difficult to understand without the deep knowledge of biology. Many researches has been made that considers the sequence data of protein to predict the behavior, structure and interactions between the proteins sequences stored in the databases. Various computational techniques have also been discussed by various researchers to enhance the interaction prediction process. An important process among all is the prediction of interactions between proteins by analyzing the sequence data. Although various efforts have been made to develop an efficient methodology for predicting the interactions, the efficiency, and performance is still a challenge for computer engineers and biologists.

Use of sequences has been done in various researches for PPI prediction. 'K-mers' is the example of such application. Although the approach has been made to move from k length patterns comparison and alignment to variable length patterns, but efficiency is still the issue. Researchers have scope to parallelize the sequence alignments and the prediction of interactions on high speed parallel organized architecture of GPU.

3. Methodology proposed

3.1 Methodology Used in Existing Algorithm

3.1.1 Apriori Algorithm

Apriori algorithm is a classical algorithm designed to work for databases containing sequence data like transactions made by customers, sequence database and others. The algorithm is used to find out the frequently occurring item sets/patterns of proteins in the database based on the fact that the item sets of frequent item sets are also frequent. The frequently occurring item sets of size K can be combined to form a K+1 size frequent item set.

3.1.2 Forming Associations between Frequent Subsequences

For every two frequent subsequences(patterns), the four possible combinations are made. i.e. the presence and absence of the subsequences can be: < p1present, p2present >;< p1present, p2absent >;< p1absent, p2present>;< p1absent, p2absent >. Where p1 represents the subsequence in protein sequence1 and p2 is the subsequence in sequence2. The values of the cells in Table 1 are incremented for all combinations of sequence pairs. These associations are shown in Table 1.

Although VLASPD provides efficient prediction of interactions among proteins. But using apriori algorithm is expensive in terms of both memory and time, and become even more complex when the candidate subsequent size increases. The research proposal tends to accelerate the task of finding variable size substrings in the sequences and finding the associations between all the possible pairs.

3.2 Parallel VLASPD(GPU Accelerated Frequent Mining and Associations Mining)

The trie data structure and vertical Bitset representation approach is used to reduce the memory consumption and the GPU implementation enhances the performance. These have been explained.

3.2.1 TrieTree and Vertical Representation of Sequence Sets

While adding an item to k-length item set to form k+1-length item set, it is considered that the portion k is same in the item sets (Figure 3). The new node is added to leaf nodes in Trie data structure by placing the sibling as the child node of the leaf node. Thus the sequences can be places as the hierarchal tree structure.

Consider the four sequences, S1-ACDEF, S2-CDEFG, S3-DEGH and S4-ACDEF. These sequences can be represented by two approaches: Horizontal and Vertical approach. Most of the recent storage techniques make use of vertical approach to store the transactions along with the items belonging to the transaction.Table 2 shows the use of vertical approach, i.e. the sequence numbers (Sequence Sets) corresponding to each particular amino acid are written in front of it. BitSet is a set of n bits, where n is the number of sequences in the consideration, where

Table 1. Forming associations between patterns

Possible Combinations	Interactions	No Interactions
$< p1, p2 >$		
$< \overline{p1}, p2 >$		
$< p1, \overline{p2} >$		
$< \overline{p1}, \overline{p2} >$		

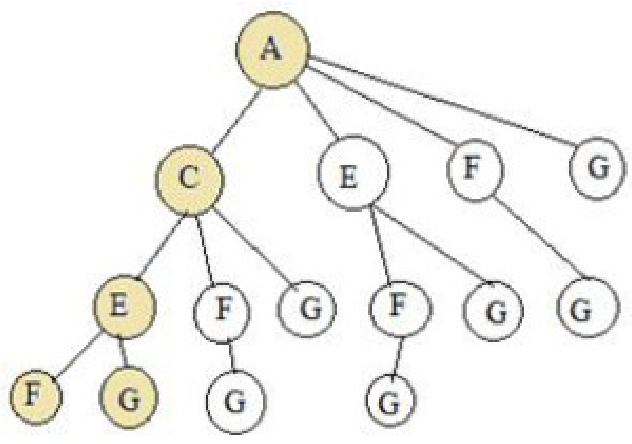

Figure 3. Trie representation of items (amino acids) for finding all possible variable length patterns in a protein. ACEFG

Table 2. Vertical representation, bitset

Amino acid alphabet	BitSet< S1; S2; S3; S4 >
A	1001
C	1101
D	1111
E	1111
F	1111
G	0110
H	0010
AC	1001
AD	1001
AE	1001
AF	1001

0 represents the occurrence and 1 represents no occurrence of the amino acid in the particular sequence. While moving from k to k+1 size item set, bitwise AND operation A AND C = 1001 AND 1101 = 1001 is performed. The BitSet combines the siblings with the leaf node of the trie to generate a new candidate set.

3.2.2 *GPU Accelerated Frequent Subsequence Mining*

The candidates are copied to GPU memory from the main memory. The support count is calculated for all the frequent variable length patterns using bitSets representation by counting the number of 1s in the bitset on GPU, as shown in flow diagram (Figure 4). The CUDA architecture involves multiple blocks, with each block comprising of threads. A thread is a basic unit. GPU processors execute threads in the groups of 32, which are called as Warps. The vertical lists are kept in 32 bits on multiple GPUs[25].

Thus, 32 bits store intersection results. This result is parallel computed on GPUs. Finally, Parallel reduction algorithm is used for counting the occurrences of number of 1s.

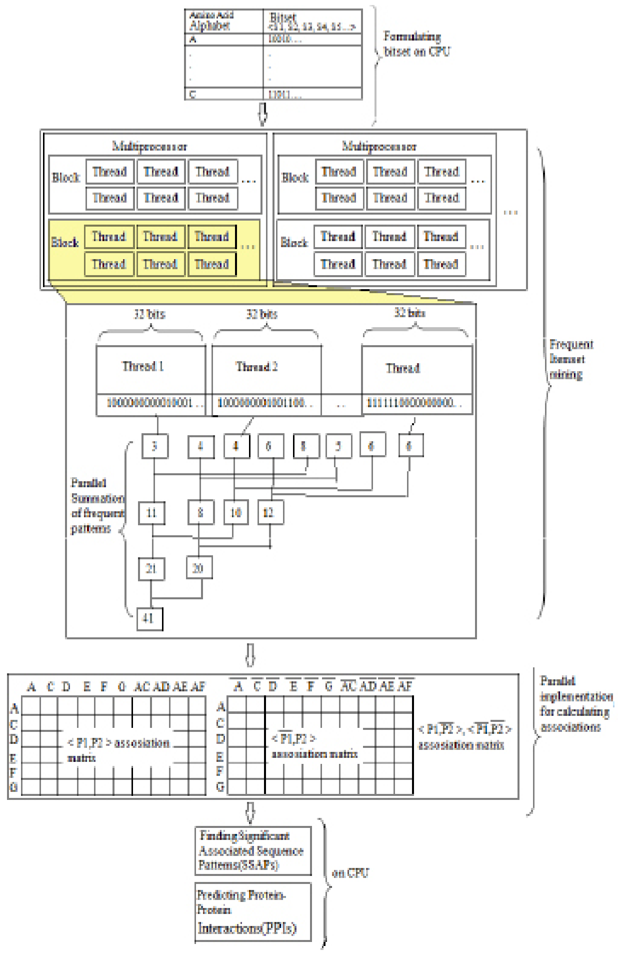
Amino Acid Alphabet
Bitset <S1, S2, S3, S4, S5...>
A
10010...
C
11011...
Formulating bitset on CPU
Multiprocessor
Block
Thread
32 bits
Thread 1
Thread 2
1000000000010001...
100000001001100...
1111110000000000...
3
4
4
6
8
5
6
6
11
8
10
12
21
20
41
Parallel Summation of frequent patterns
Frequent Itemset mining
A C D E F G AC AD AE AF
< P1,P2 > association matrix
< P1,P2 >, < P1,P2 > association matrix
Parallel implementation for calculating associations
Finding Significant Associated Sequence Patterns(SSAPs)
Predicting Protein-Protein Interactions(PPIs)
on CPU

Figure 4. GPU implementation of VLASPD.

3.2.3 *Finding Associations on GPU*

Matrices are used to manage the task of associations. The task of matrix filling is implemented on GPUs as associating 1-length subsequence to all possible k-length subsequences on GPU1(k∈ 1 to maximum length pattern),2-length subsequence on GPU2, k-length subsequence on GPU$_k$, and k+1 length subsequence on GPU$_{k+1}$. Thus parallelizing the calculations of associations on GPU can be possible.

The remaining steps for finding significant associations and interaction predictions are implemented as per VLASPD algorithm[7].

3.3 Frequent Patterns using FP-growth Algorithm

The apriori algorithm, since it basically works on generate-and-test approach, first generates the candidates, and then tests the frequency of the candidates. The candidate generation step makes apriori algorithm complex in terms of space as well as time. The support count calculation is computationally expensive and involves multiple database scans (I/O expensive). GPU based apriori reduces the complexity by parallelizing candidate generation. But another recent approach 'FP-Growth' provides higher performance than GPU based apriori since the candidate generation step is completely removed[26].

FP-Growth uses two-step approach to generate the frequent patterns without the generation of candidates. The two steps followed are: (i) Build an FP-Tree. (ii) Extract frequent patterns from FP-Tree.

3.3.1 FP-Tree Construction

The step reframes the entire protein database into a FP-Tree. Two sub steps performed are:

Sub step 1: Scan the protein sequences database(Table 3) to find the support count of each amino acid(Table 4). The less frequent amino acids are discarded. The remaining frequent amino acids are arranged in order of their frequency (Table 5). Consider the example of sequences:

Sub step 2: FP-Tree is constructed as a tree with amino acids placed as nodes, each node has a counter associated with it. The initial node is kept as null. FP-Growth reads a single sequence at a time and place it on the tree by tracing the path which it matches (Figure 5 Figure 6). The sequences with common prefixes share paths, the counters are incremented.

The FP-tree has less size as compared to the original database, since many sequences share has same amino acids in them. We can say that FP-Growth compresses the database into FP-Tree.

Table 3. Sequences in an example database

Sequence id	Sequence
S1	ACF
S2	CE
S3	CD
S4	ACE
S5	AD
S6	CD
S7	AD
S8	ACDF
S9	ACD

Table 4. Calculating frequency of each amino acid

Item	Frequency	Priority
A	6	2
C	7	1
D	6	3
E	2	4
F	2	5

Table 5. Amino acids arranged in order of their priority

Sequence id	Rearranged Sequence
S1	CAF
S2	CE
S3	CD
S4	CAE
S5	AD
S6	CD
S7	AD
S8	CADF
S9	CAD

3.3.2 Extracting Frequent Patterns from FP-Tree

For node F, the paths are: {C,A,F} and {C,A,D,F}. Paths without prefix are: {C,A:1} and {C,A,D:1}. Frequent set generated is: {C,A : 2}>=2. Frequent patterns are: {A,F} {C,F},{C,A,F}. Similarly, by considering support count as 2, the frequent patterns are {C,E},{C,A,D},{C,A}.

3.4 Proposed Work

The methodology proposed performs the hybridization of the two algorithms, Apriori and FP-Growth algorithm, and implements the parallel algorithm on GPU. The

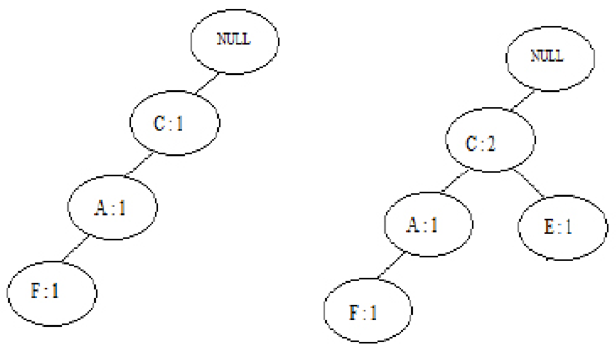

Figure 5. (**a**) FP-Tree after reading sequence 1 (**b**) FP-Tree after reading sequence 2.

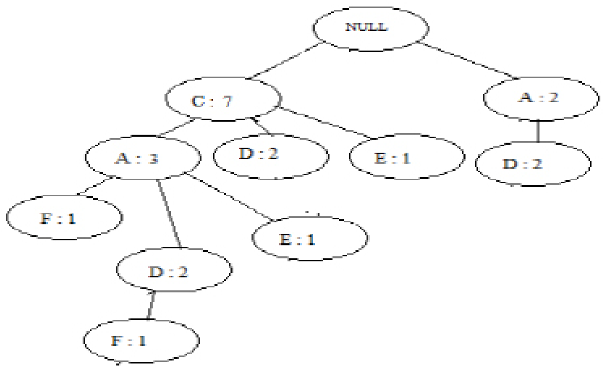

Figure 6. Final FP Tree after reading S9.

space and time complexity, and IO access of the existing techniques are challenged.

4. Expected Outcomes

The research project is expected to provide a new architecture for an efficient prediction of interactions between all the possible pairs of protein sequences in the protein database with the use of CUDA architecture and GPUs. The parallel task on GPU will reduce the time required for finding the frequent patterns in proteins, and the associations among them. Thus the overall implementation time of the VLASPD algorithm will be reduced.

5. Summary and Conclusion

Protein Protein Interactions (PPIs) have a major impact on the biological processes. Even though, various methodologies have been developed for predicting PPIs, the practicality of most existing methods is limited because they need information about protein homology or requires a tedious task of collecting biological information. In the work, we parallelized an existing methodology for PPI prediction that used only the information of protein sequences and their frequency in the database. The paper presents a parallel VLASPD algorithm, which parallelize two important steps of VLASPED: Frequent subsequences mining and finding the associations between the frequent patterns. This parallel method uses highly parallel architecture of GPU that uses CUDA programming framework to improve the performance and efficiency by reducing the implementation time.

6. References

1. Luscombe NM, Greenbaum D, Gerstein M. What is bioinformatics? A proposed definition and overview of the field. Methods Information Medical. 2001; 40(4):346–58.
2. Kidera A, Konishi Y, Ooi T, Scheraga HA. Relation between sequence similarity and structural similarity in proteins. Role of important properties of amino acids.Journal of Protein Chemistry.1985; 4(5):265–97.
3. Pearson WR. Searching protein sequence libraries: Comparison of the sensitivity and selectivity of the Smith-Waterman and FASTA algorithms.Genomics. 1991; 11(3):635–50.
4. Fogg CN, Kovats DE. Computational Biology: Moving into the future one click at a time. PLOS Computing Biology. 2015; 11(6).
5. Hu L, Chan K. Discovering variable-length patterns in protein sequences for protein-protein interaction prediction. IEEE Transaction on Nanobioscience. 2015; 14(4):409–16.
6. Luebke D, Humphreys G. How GPUs work. Computer (Long Beach Calif). 2007; 40(2):96–100.
7. Ben-Hur A, Noble WS. Kernel methods for predicting protein–protein interactions.Bioinformatics. 2005; 21(suppl 1):38–46.
8. Shen J, Zhang J, Luo X, Zhu W, Yu K, Chen K. Predicting protein–protein interactions based only on sequences information.Proceeding of National Acadamy Science. 2007; 104(11):4337–41.
9. Liu Y, Huang W, Johnson J, Vaidya S. GPU accelerated smith-waterman. Computational Science–ICCS 2006; 2006. p. 188–95.
10 Manavski SA, Valle G. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment.BMC Bioinformatics. 2008; 9(2):1–9.
11. Striemer GM, Akoglu A. Sequence alignment with GPU: Performance and design challenges. 2009 IPDPS IEEE International Symposium on Parallel and Distributed Processing; 2009. p. 1–10.

12. Liu Y, Maskell DL, Schmidt B. CUDASW++: Optimizing Smith-Waterman sequence database searches for CUDA-enabled graphics processing units. BMC Research Notes. 2009; 2(1):73.
13. Ligowski Ł, Rudnicki W. An efficient implementation of Smith Waterman algorithm on GPU using CUDA, for massively parallel scanning of sequence databases. 2009 IPDPSIEEE International Symposium on Parallel and Distributed Processing; 2009. p. 1–8.
14. Rudnicki WR, Jankowski A, Modzelewski A, Piotrowski A, Zadrożny A. The new SIMD implementation of the Smith-Waterman algorithm on Cell microprocessor. FundamInformaticae. 2009; 96(1–2):181–94.
15. Farrar M. Striped Smith–Waterman speeds database searches six times over other SIMD implementations. Bioinformatics. 2007; 23(2):156–61.
16. Khajeh-Saeed A, Poole S, Perot JB. Acceleration of the Smith–Waterman algorithm using single and multiple graphics processors. Journal of Computational Physics. 2010; 229(11):4247–58.
17. Liu Y, Schmidt B, Maskell DL. CUDASW++ 2.0: enhanced Smith-Waterman protein database search on CUDA-enabled GPUs based on SIMT and virtualized SIMD abstractions. BMC Research Notes. 2010; 3(1):93.
18. Blazewicz J, Frohmberg W, Kierzynka M, Pesch E, Wojciechowski P. Protein alignment algorithms with an efficient backtracking routine on multiple GPUs. BMC Bioinformatics. 2011; 12(181):1–17.
19. Hasan L, Kentie M, Al-Ars Z. GPU-accelerated protein sequence alignment. Engineering in Medicine and Biology Society, EMBC, Annual International Conference of the IEEE; 2011. p. 2442–6.
20. Hains D, Cashero Z, Ottenberg M, Bohm W, Rajopadhye S. Improving CUDASW++, a parallelization of Smith-Waterman for CUDA enabled devices. IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW); 2011. p. 490–501.
21. Liu Y, Wirawan A, Schmidt B. CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions. BMC Bioinformatics. 2013; 14(1):117.
22. Lee ST, Lin CY, Hung CL. GPU-based cloud service for smith-waterman algorithm using frequency distance filtration scheme. Biomed Research International. 2013; 2013:8.
23. Liu Y, Hong Y, Lin C-Y, Hung C-L. Accelerating Smith-Waterman Alignment for protein database search using frequency distance filtration scheme based on CPU-GPU collaborative system. International Journal of Genomics. 2015; 2015:12.
24. Houtgast EJ, Sima VM, Bertels K, Al-Ars Z. An FPGA-based systolic array to accelerate the BWA-MEM genomic mapping algorithm. International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation (SAMOS); IEEE. 2015. p. 221–7.
25. Huang L, Wu C, Lai L, Li Y. Improving the mapping of Smith-Waterman sequence database searches onto CUDA-enabled GPUs. BioMed Research International. 2015; 2015:10.
26. Fang W, Lu M, Xiao X, He B, Luo Q. Frequent itemset mining on graphics processors. Proceedings of the fifth international workshop on data management on new hardware. ACM; 2009. p. 34–42.