ISSN:0975-9646

Performance Efficient DNA Sequence Detection on GPUUsing Parallel Pattern Matching Approach

Rahul Shirude[1] Valmik B. Nikam[2] B.B. Meshram[3]

[1, 2, 3] *Department of Computer Engineering and Information Technology*
Veermata Jijabai Technological Institute
Mumbai, India

***Abstract*— Bioinformatics is THE field of science which applies computer science and information technology to the problems of biological science. One of the most useful applications of bioinformatics is sequence analysis. Sequence analysis, which is the process of subjecting a DNA, RNA to any wide range of analytical approaches, involves methodologies like sequence alignment and searches against biological databases. For the analysis DNA sequences are stored in databases for easy retrieval and comparison. Frequency of pattern occurrence in database may predict the intensity of the disease. When the sequence database is huge, matching a pattern is very time consuming task. This fact leads to the need of utilizing latest complex and expensive hardware like GPU.**

In this paper, we propose a Parallel string matching algorithm using CUDA (Compute Unified Device Architecture). The focus of the research is the design and implementation of an algorithm by utilizing GPU cores optimally. Our algorithms finds correct matches and experimental results show very high performance gain over the sequential approach.

***Keywords*— Bioinformatics; Matching; Pattern; DNA sequence; Sequence Analysis;CUDA.**

I. INTRODUCTION

As the growth rate of biological sequence databases increased, the demand for advanced and high performance computational method for comparing and searching biological sequences have also increased. In DNA sequence alignment [14], the performance of comparison and alignment affect a lot of application processes such as vaccines design, drugs, disease detection and curing method. Hence with the high performance and high sensitivity DNA sequences alignment or comparison the vaccines, drugs, disease detection and disease curing method can be designed and defined in a faster way. To satisfy this need, high performance and sensitive DNA sequence matching algorithms are very important for research and application of molecular biology today. Biological sequence alignment is a computationally expensive application in the field of bioinformatics and computational biology as its computing and memory requirements grow quadratic ally with the size of the databases. It aims to find out whether two or more biological sequences are related or not.

The problem of exact string matching is to find all occurrences of pattern 'P' of size 'm' in the text string 'T' of size 'n'. Let P = {p1, p2, p3 ,….,pm} be a set of patterns of m characters and T={t=t1,t2,t3…,tn} in a text of n characters which are strings of nucleotide sequence characters from a fixed alphabet set called Σ= {A, C, G, T}[2]. Let T be a large text consisting of characters in Σ. In other words T is an element of Σ*. The problem is to find all the occurrences of pattern P in text T. It is an important application widely used in data filtering to find selected patterns, in security applications, and is also used for DNA searching[1].Pattern matching focuses on finding the occurrences of a particular pattern in a text file. The problem in pattern discovery is to determine how often a candidate pattern occurs, as well as possibly some information on its frequency distribution across the sequence/text. In general, a pattern will be a description of a set of strings, each string being a sequence of symbols. Hence, given a pattern, it is usual to ask for its frequency, as well as to examine its occurrences in a given sequence/text. The biologists often queries new discoveries against a collection of sequence databases such as EMBL[22], GENBANK[15] and DDBJ[16] to find the similarity sequences. As the size of the data grows it becomes more difficult for users to retrieve necessary information from the sequences. Hence more efficient and robust methods are needed for fast pattern matching techniques. We have proposed an accelerated approach of string and pattern matching algorithms to find out a particular sequence or pattern in the given DNA database using parallel programming approach. This can be accomplished by parallelization technique on GPU using CUDA programming model. Pattern matching algorithms have main objectives such as:

Design and implement sequential algorithm for pattern matching for DNA sequence analysis.

To parallelize the developed sequential algorithm.

To analyze the scalability and optimizing accordingly.

The rest of the paper is organized as follows; Section II is Literature survey, Section III is Proposed Approach, Section IV is Experimental System Requirements and we make some concluding remarks in Section V.

II. LITERATURE SURVEY

A. Study of pattern matching algorithms

The literature describes various traditional pattern matching methodologies like Naive Brute force, Boyer Moore, Knuth Morris Pratt and Dynamic algorithms along with their performance issues when applied for sequence analysis. Pattern matching is used in various processes like DNA sequencing, Intrusion Detection System.

1) Naive Brute force

It is one of the simplest algorithms having complexity O(mn). In this, First character of pattern P(with length m) is aligned with first character of text T (with length n).Then scanning is done from left to right. As shifting is done at each step it gives less efficiency. [17]

2) Boyer-Moore Algorithm

It performs larger shift-increment whenever mismatch is detected. It differs from Naïve in the way of scanning. It scans the string from right to left; unlike Naive i.e. P is aligned with T such that last character of P will be matched to first character of T. If character is matched then pointer is shifted to left to very rest of the characters of the pattern. If a mismatch is detected at say character c, in T which is not in P, then P is shifted right to m positions and P is aligned to the next character after c. If c is part of P, then P is shifted right so that c is aligned with the right most occurrence of c in P. The worst complexity is still O (m+n) [11].

3) Knuth-Morris-Pratt

This algorithm is based on automaton theory. Firstly a finite state automata model M is being created for the given pattern P. The input string T with Σ= {A, C, T, G} is processed through the model. If pattern is present in text, the text is accepted otherwise rejected. But the only disadvantage of the KMP algorithm [3] is that it doesn't tell the number of occurrences of the pattern[12].

4) Dynamic programming Algorithms

Dynamic programming is the oldest and mostly used algorithm. Basically Needleman Wunsch and Smith waterman algorithm[18] come under this approach. These are much more complex than the exact pattern matching. It involved solving successive recurrence relations recursively i.e. smaller problems are solved in succession to solve the main problem.

a)Smith-Waterman (local alignment)[18]

• Accuracy: good with gapped pairs
• Processing: Computationally expensive O (N^2) and with trace-back a lot of ^{memory} is required; this is slow
• Limitations: indexing to find targets is required.

b)Needleman-Wunsch (global alignment)[18]

• Good for small genomes and long matching alignments
• Processing: Computationally expensive O (N^2) Talk today showed novel pruning technique for in large matches.
• Limitations: requires hard left hand bound known query and target size.

B. Related work

1)Existing programs on CPU

• BWA [6]
• BFAST[7]
• Mosaik[8]
• BLAST[9]

Problem with existing programs is that they are slow, less accurate, And/or require large memory. Expensive hardware is required to run these programs. Cheaper hardware is more desirable. So GPUs are a good alternative.

2)Current GPU based tools

a) Smith-Waterman implementations

- CUDASW++: is a bioinformatics software for Smith-Waterman protein database searches that takes advantage of the massively parallel architecture of GPUs to perform sequence searches. It has drawback as it deals only with protein sequence alignment only[19].
- SeqNFind: The SeqNFind Smith Waterman tool allows examination of local alignments at every location within a genome. It has some drawback as it is Commercial and Need to buy along with hardware[13].
- MuMmerGPU: is a high-throughput DNA sequence alignment program that runs on Nvidia G80-class GPUs. It aligns sequences in parallel on the video card to accelerate the widely used serial CPU program MuMmer. It shows no gapped alignment[20].

C. GPU/CUDA Architecture

CUDA architecture is different from the general computer system. To choose better architecture is important for improving performance. Multiple GPUs also can be utilized for high performance gain [10]. As we consider hardware architecture of CUDA supported GPU's, SM (Streaming Multiprocessor) is used for thread execution. Each SM contains 8, 16 or 96 stream processors and support up to 8 or 16 blocks of concurrently executing threads. Warp contains number of threads in a block that executes simultaneously. Each SM manages a number of specified warps; therefore the maximum number of threads in a SM is number of warps multiplied by number of threads.[23] Blocks are grouped into grids; the kernel function executes grids of blocks of threads. For accessing global memory from GPU require longer access latency. To avoid this longer latency a portion of global memory can be bound as a texture memory which is used for catching and occurs when cache miss occurs. Usually texture memory has good performance than global memory. GPU platform also supports some fast memory. One of that is constant memory which is read only memory and is used for catching. Another memory is shared memory which is owned by each SM and performs both read and writes operations [4]. Figure.1. shows the memory structure of the CUDA[5].

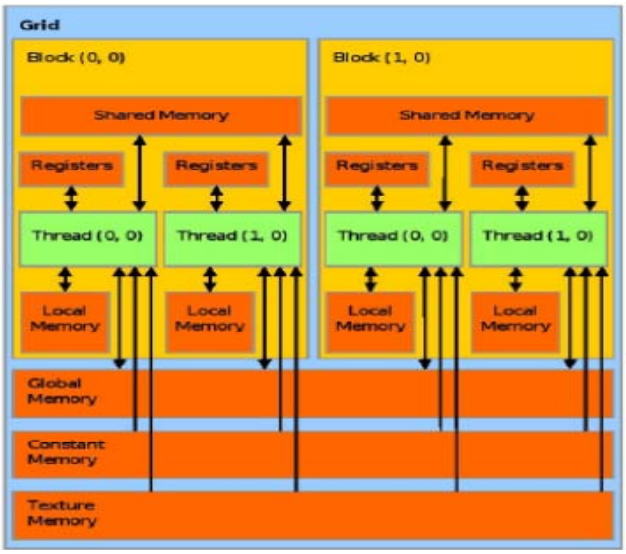

Fig. 1. CUDA memory model

Shared memory is the memory located on multiprocessors of the device itself and is shared by all threads of a thread block. This is the memory which can be accessed by both host and device. The data which is stored in main memory must be copied from main memory to global memory by using CUDA memory copy functions, if it should be accessed by device. All the threads have its local memory. General global memory is available up to 6 GB per GPU having bandwidth up to 180 GB/s for the Tesla products. If the data is read only data and if the space is not available for storing the data then the texture memory is used for this purpose. In constant memory constants are immutable and not be written by kernel, even in dynamic parallelism kernel launches. Constant memory variables are globally visible to all kernels. Texture memory access and writes to global memory object that relates with texture memory objects between parent and children.

1) Thread Hierarchy:

For convenience, threadIdx is a 3-component vector, so that threads can be identified using a one-dimensional, two-dimensional, or three-dimensional *thread index*, forming a one-dimensional, two-dimensional, or three-dimensional thread block. This provides a natural way to invoke computation across the elements in a domain such as a vector, matrix, or volume. The index of a thread and its thread ID relate to each other in a straightforward way: For a one-dimensional block, they are the same; for a two-dimensional block of size (D_x, D_y),the thread ID of a thread of index (x, y) is $(x + y\ D_x)$; for a three-dimensional block of size (D_x, D_y, D_z), the thread ID of a thread of index (x, y, z) is$(x + y\ D_x + z\ D_x\ D_y)$.There is a limit to the number of threads per block, since all threads of a block are expected to reside on the same processor core and must share the limited memory resources of that core. On current GPUs, a thread block may contain up to 1024 threads. However, a kernel can be executed by multiple equally-shaped thread blocks, so that the total number of threads is equal to the number of threads per block times the number of blocks. Blocks are organized into a one-dimensional, two-dimensional, or three-dimensional *grid* of thread blocks as illustrated by Figure 2. The number of thread blocks in a grid is usually dictated by the size of the data being processed or the number of processors in the system, which it can greatly exceed.

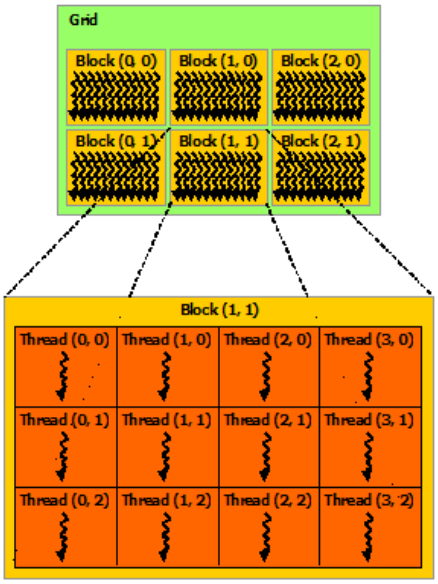

Fig.2. Grid of Thread Blocks

Each block within the grid can be identified by a one-dimensional, two-dimensional, or three-dimensional index accessible within the kernel through the built in blockIdx variable. The dimension of the thread block is accessible within the kernel through the built in blockDim variable.

III. Proposed Approach

In order to apply the maximum level of parallelization and to improve performance, a brute force method is chosen to test the parallel approach. This approach is applied for the process of disease diagnosis by matching input gene pattern with DNA database text file and draw conclusions from number of occurrences or matching of input gene pattern. There are some objectives to be followed for completion of research and these objectives are as follows:

- Analyse different serial algorithms for pattern matching and their performance ratios.
- Build scalable parallel pattern matching algorithm for DNA sequencing.
- Measure the GPU and CPU performance differences in terms of processing time and generate resultant graph. Various input gene patterns are match with DNA database file and verified according to count of occurrences of pattern into DNA database file. Threads are run according to size of database file.

A. Naive Brute Force String Matching Algorithm:

Naive Brute Force String Matching Algorithm is a basic algorithm that takes a string S, of size n, and a pattern P, of size m, and scans the first n-m elements of the string from left to right with the pattern, looking for matches. Basically the algorithm considers all the possible starting positions of the pattern(P) for j=0 to n-m. Then for every starting position (j) the pattern(P) must exactly match S for the next consecutive m-1 positions. The result of the algorithm is the set I containing all of the starting positions in S where P exactly matches the string S(using indices starting at 1)[21].

As an example consider P=aba and S=acbababa, which has the following iterations in the Naïve String Matching Algorithm:

P=aba , S=acbababa

Iterations:

j=0 **a**cbababa
aba

j=1 acbababa
aba

j=2 acbababa
aba

j=3 acbababa
aba

j=4 acbababa
aba

j=5 acbababa
aba

The output from the Naive String Matching Algorithm would be the set I={3+1,5+1}={4,6}(note the plus 1 results from sequence indexes starting at 1 not 0) since for all other values of j the pattern P did not match exactly.

The sequential form of algorithm consists of function, where it attempts to match pattern of text by scanning text from left to right. In sequential code, a single thread is conducting the search and when it finds a match the algorithm will output to console the position it was found.

In CUDA version, N threads could be conducting the same search. Each of the N threads attempt to scan for a match of the text, in parallel and when it discovers a match an array for storing the found indices will be updated. Each CUDA thread can potentially and possibly read each character and obtain a match , in the event that the pattern follows one another in string

B. Proposed Model

CUDA is a parallel computing architecture developed by NVIDIA. The aim is to provide a programming framework for general purpose computations on Graphics Processing Units. CUDA programming model assumes that each CUDA thread has its own local memory and is running on one of the stream processors of the GPU multiprocessor sharing on-chip memory with the other threads running on the multiprocessor.

The suggested parallel computational model uses pattern matching algorithm distributed between the threads of the GPU kernels. Each thread calculates the matching of input pattern with stored database file (character array). The flow chart is given in Figure 3.

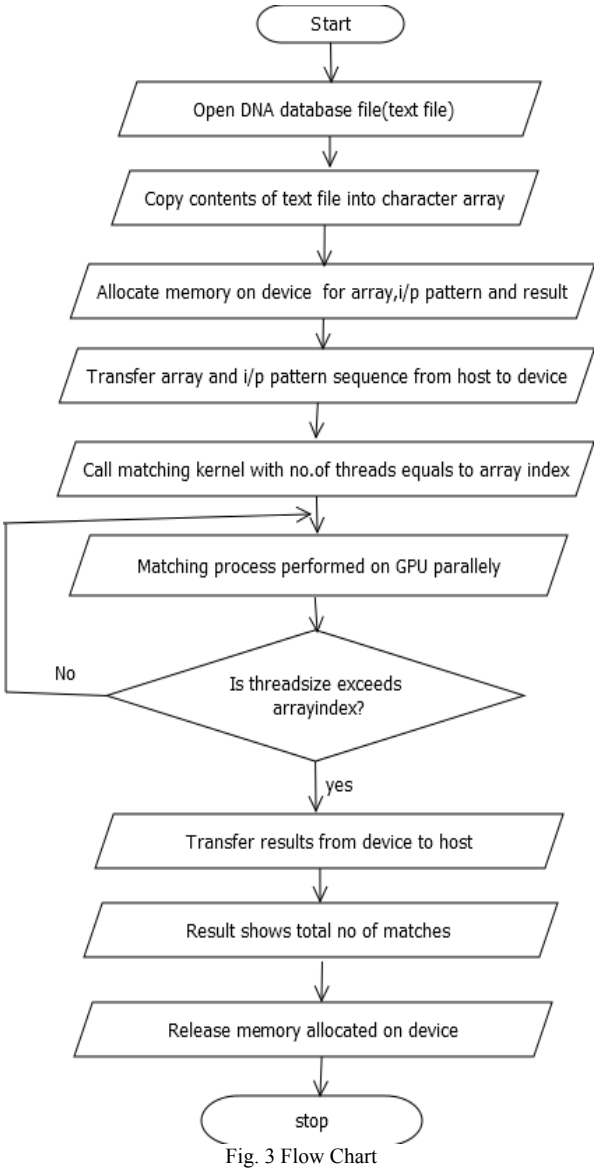

Fig. 3 Flow Chart

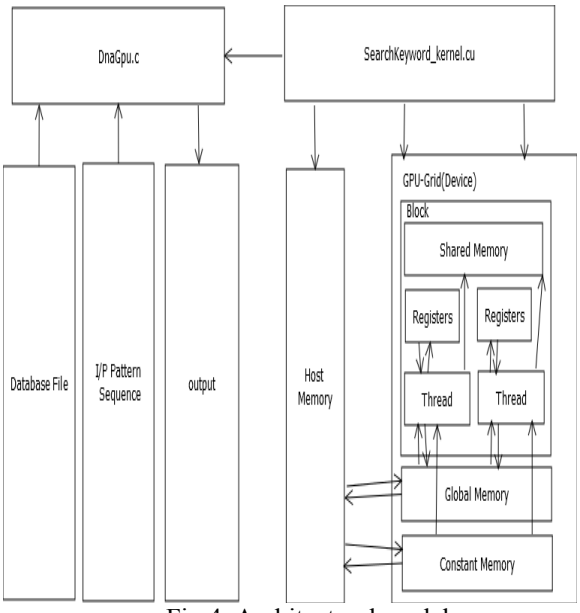

Fig.4. Architectural model

The proposed (Architectural) model shown in Figure 4 include following modules:

- Main module(DnaGpu.c) in C
- Kernelcode (Searchkeyword_kernel.cu) using CUDA.

Main module (DnaGpu.c) does file handling operations and also calling of the kernel function. The database file is first stored into the character Array. And index of array shows size of database text file. The database text array and input pattern is passed to GPU device as well as size of database file is passed to GPU for processing. Numbers of threads passed to kernel are equals to size of text array passed to GPU. So each thread does processing parallel with input pattern and calculates number of occurrences of pattern in database text file. Then passed output in the form of total no. of matching to Host (CPU).

C. Flow Chart

At the very beginning of the CUDA code's execution, code is compiled just like other c code. Its primary execution takes place in CPU.As the execution started all non-kernel functions getting executed on CPU and the execution of kernel code is being transferred to GPU. This way we get parallel execution on CPU and GPU. Once the memory transfer between CPU to GPU is done, without any impediments the rest of the execution is carried well otherwise execution will be halted. Pattern matching gives out the search results for presence of specific input pattern in DNA sequence database. The simplest brute force technique is used for the matching so as to cope up with complexity and to prevent possible overhead occurring due to parallelization. In order to reduce searching time matching is carried out parallel that reduces the search time with accurate retrieval. Data is collected from well-known database NCBI and other genome projects.

IV. Experimental System Requirements

A. Experimental System Requirements

Experiments were performed on the machine with Intel Xeon E5-2650 processor and Nvidia Tesla K20 GPU. The Nvidia Tesla K20 graphics processing unit (GPU) active board is PCI Express, comprising of a single GK 110 GPU. The Tesla K20 active accelerator is designed for workstation and offer total of 5 GB GDDR-5 on-board memory and supports PCI Express Gen2. The Tesla K20 GPU has 2496 processor cores and also has 3.52 Teraflops single and 1.17 Teraflops double precession floating point units. Nvidia profiler is used to measure kernel execution time and data transfer time. The GTX titan graphics card has the following features:-

- Compute capability - 3.5.
- Threads per block - 1024.
- Shared memory per block - 48KB.
- Registers per block - 65536.
- Warps per multiprocessor - 64.
- Blocks per multiprocessor - 16.
- Global memory bandwidth - 275.02 GB/s / size= 5.99GB.
- Constant memory - 64KB.

V. Conclusion

In this paper we have implemented an efficient parallel Pattern Matching Algorithm by utilizing GPU cores optimally on CUDA. The proposed technique enhances the comparison time and performance when compared with sequential approach of algorithm on CPU. From the obtained results, that means from count of number of occurrences of given pattern in DNA database file we can conclude that whether an individual has chances of getting disease or not in future. The research has been done On GPU using CUDA programming model, accelerating the pattern matching process. The proposed algorithm will take advantages of parallel computing and give better time and money saving approach for disease detection. Our algorithms finds correct matches and experimental results show very high performance gain over the sequential approach.

References

[1] Bhukya R., Somayajulu DVLN, "Exact multiple pattern matching algorithm using DNA sequence and pattern pair", International Journal of Computer Applications (0975 – 8887) Volume 17– No.8, March 2011

[2] Bhukya R., Somayajulu DVLN, "2-Jump DNA Search Multiple Pattern Matching Algorithm", IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 3, No. 1, May 2011.

[3] S. Rajesh , S.Prathima, Dr.L.S.S.Reddy, "Unusual Pattern Detection in DNA Database Using KMP Algorithm",2010 International Journal of Computer Applications(0975-8887) Volume 1 – No.22

[4] http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

[5] Kirk,D., Hwu,W.," Programming Massively Parallel Processor" second edition.

[6] Heng Li., Richard Durbin "Fast and accurate short read alignment with Burrows–Wheeler Transform " Vol. 25 no. 14 2009, pages 1754–1760 doi:10.1093/bioinformatics/btp324 May,2009

[7] Homer N, Merriman B, Nelson SF (2009) " BFAST: An Alignment Tool for Large Scale Genome Resequencing." PLoS ONE 4(11): e7767. doi:10.1371/journal.pone.0007767

[8] Lee W-P, Stromberg MP, Ward A, Stewart C, Garrison EP, et al. (2014) "MOSAIK: A Hash-Based Algorithm for Accurate Next-Generation Sequencing Short-Read Mapping." PLoS ONE 9(3): e90581. doi:10.1371/journal.pone.0090581

[9] Stephen F. Altschul, Warren Gish, Webb Miller, Eugene W. Myres, David J. Lipman "Basic local alignment search tool" Journal of Molecular Biology Volume 215,Issue 3,5 October 1990,Pages 403-410 DOI:10.1016/S0022-2836(05)80360-2

[10] Urvesh Devani, V. B. Nikam, B.B. Meshram (2014) "On the fly Porn Video Blocking using Distributed Multi-GPU and Data Mining Approach", International Journal of Distributed and Parallel Systems, July 2014.

[11] R.S. Boyer, J.S. Moore, "A fast string searching algorithm," *Communication of the ACM*, Vol. 20, No. 10, 1977, pp.762–772.

[12] KNUTH, D. E, MORRIS JR J. H , PRATT V. R,"Fast pattern matching in strings", In the procd. Of SIAM J.Comput.Vol. 6, 1, pp. 323–350, 1977

[13] http://www.atlab.com/docs/ATLSeqNFinddatasheet.pdf

[14] Mount D. Bioinformatics: Sequence and Genome Analysis, Cold Spring Harbor Laboratory(CSHL) Press,2004.

[15] http://www.ncbi.nlm.nih.gov/genbank

[16] http://www.ddbj.nig.ac.jp

[17] http://codeaspirant.wordpress.com/2013/05/20/brute-force-naive-approach-to-string-searching

[18] http://fenchurch.mc.vanderbilt.edu/lab/bmif310/2012/2-D-Comparing-Sequences-NW-and-SW.pdf

[19] http://cudasw.sourceforge.net/homepage.htm#latest

[20] http://www.biomedcentral.com/1471-2105/8/474

[21] Parida, Laxmi (2008) *Pattern Discovery in Bioinformatics: Theory & Algorithms* Boca Raton: Chapman & Hall/CRC pg. 139-182,183-212

[22] Stoesser, G., Baker, W., van den Broek, A.E., Camon,E., Hingamp,P., Sterk,P. and Tuli,M.A.(2000) The EMBL Nucleotide Sequence Database. *Nucleic Acids Res*., 28, 19–23

[23] Sangale,A.,Devani,U.,Nikam,V.,Meshram,B.,"Implementing Adaptiv and Dynamic Data Structures using CUDA Parallelism",ICAETR ,2014IEEE International Conference on ,Auguest 1-2 ,2014.

AUTHOR'S INFORMATION

Rahul R. Shirude is Bachelor of Engineering (Computer Engineering) from University of Pune, Pune and pursuing his Master of Technology in Information Technology (with specialization in Software Engineering) from VJTI, Matunga, Mumbai, Maharashtra state. He is working on "Parallel approach of String and Pattern matching algorithm in Bioinformatics on GPU using CUDA" for as a part of his thesis. He has been a part of various technical workshops and research meet ups as a student and also as a committee member. His research interests include Parallel Processing, High Performance Computing, Mobile Computing and Cloud Computing,. He worked as a Technical Officer for CSI Event 'Tantranaad' held at VJTI. He has also completed certification in cloud computing i.e.Salesforce.com (Dev-401).

Valmik B Nikam is Bachelor of Engineering (Computer Science and Engineering) from Government College of Engineering Aurangabad, Master of Engineering (Computer Engineering) from VJTI, Matunga, Mumbai, Maharashtra state, and pursuing PhD in Computer Department of VJTI. He was faculty at Dr. Babasaheb Ambedkar Technological University, Lonere. He has 12 years of academic experience and 5 years of administrative experience as a Head of Department. He has one year of industry experience. He has attended many short term training programs and has been invited for expert lectures in the workshops. Presently he is Associate Professor at deparment of Computer Engineering & Information Technology of VJTI, Matunga, Mumbai. His research interests include Scalability of Data Mining Algorithms, Data Warehousing, Big Data, Parallel Computing, GPU Computing, Cloud Computing. He is member of CSI, ACM, IEEE research organizations, and also a life member of ISTE. He has been felicitated with IBM-DRONA award in 2011.

B.B.Meshram is a Professor and Head of Department of Computer Engineering and Information Technology, Veermata Jijabai Technological Institute, Matunga, Mumbai. He is Ph.D. in Computer Engineering. He has been in the academics & research since 20 years. His current research includes database technologies, data mining, securities, forensic analysis, video processing, distributed computing. He has authored over 203 research publications, out of which over 38 publications at National, 91 publications at international conferences, and more than 71 in international journals, also he has filed two patents. He has given numerous invited talks at various conferences, workshops, training programs and also served as chair/co-chair for many conferences/workshops in the area of computer science and engineering. The industry demanded M.Tech program on Network Infrastructure Management System, and the International conference "Interface" are his brain childs to interface the industry, academia & researchers. Beyond the researcher, he also runs the Jeman Educational Society to uplift the needy and deprived students of the society, as a responsibility towards the society and hence the Nation.