Journal of Theoretical and Applied Information Technology
20th May 2014. Vol. 63 No.2
© 2005 - 2014 JATIT & LLS. All rights reserved·

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

PARALLEL MODEL AND SCHEDULING TECHNIQUE FOR SPACES COMPLEXITY AND SYNCHRONIZATION PROBLEMS IN SEQUENCES ALIGNMENT

[1]MANHAL ELFADIL ELTAYEEB, [2]MUHAMMAD SHAFIE ABD LATIF, [3]ISMAIL FAUZI ISNIN

[1,2,3]Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, Johor, Malaysia

E-mail: [1]manhalus@gmail.com, [2]shafie@utm.my, [3]ismailfauzi@utm.my

ABSTRACT

Biologists are confusing with the huge amount of data resulting from conformations of DNA and protein sequences. In an earlier stage, a dot-plot method is used to identify new sequences. It is based on comparing sequences in a level of graphical illustration to detect similar locations of sequences. However, for long sequences this method is impractical. Furthermore, Improvement method using sequential machine adopted by Needleman-Wunsch (NW) and Smith-Waterman (SW) algorithms, where sequences set in a matrix with scoring system and optimal alignment via dynamic programming method is achieved. Unfortunately, these algorithms suffer from time and space complexity. An alternative approach is necessary to compare long sequences in a reasonable time with respect to memory restrictions. In this paper, we developed a new parallel model with implementing scheduler-worker paradigm and a scheduling technique. Our model is based on Bulk Synchronous Parallelism (BSP) model, where each worker has its own distributed memory and accomplish selected number of blocks. Using X86-based PC with eight logical processors we are able to compare sequences range from 411 KBP to 4 MBP in $O(\frac{m+\frac{n}{w}}{w})$ space and linear communication complexity.

Keywords: *DNA and Protein, Sequences Comparisons, Parallel Model, Memory and Communications Complexities*

1. INTRODUCTION

In recent years, there had been an increasing interest in computational biology problems such as predicting the structure and functions of newly DNA or proteins. However, the fundamental issue in computational biology is aligning similar DNA or protein sequences in order to reveal or predict functional, structural, and evolutionary analogies between sequences. Sequence alignment is a problem of matching similar regions between biological sequences. Pairwise sequence alignment is dedicated to aligning two sequences, while, in multiple sequences more than two sequences are aligned. Global sequence alignment is used to compare sequences as a whole, while local alignments is appropriate for detecting specifically conserved regions.

A central problem of sequence alignment is a memory restriction in comparing long sequences. Numerous experiments have been established for solving space complexity in the sequence alignment. [1], used Dynamic Programming (DP) techniques to search for optimal alignment in global sequence alignment. It is extended to local sequence alignment by [2]. DP algorithms guarantee an accurate result with optimal alignment. However, for long sequences length these methods tend to be very slow and expensive. Heuristic algorithms such as FASTA [3] and BLAST [4] were developed to accelerate sequences comparisons while striving to keep sensitivity as best as possible. Both FASTA and BLAST are much faster but produce inaccurate results [5]. Furthermore, the computational complexity of both FASTA and BLAST is *O(MN)*, while the space complexity for FASTA is *O(MN)* and for BLAST is slightly higher than all other algorithms, it is *O(20w+MN)*, see Table 1. This paper focuses on implementing space complexity with scheduling techniques for local sequence alignment.

For a number of years, numerous investigations proposed to address the lack of computing power in space complexity problems ranging from incorporating inventing new algorithms into the ROM of a specialized chip to adopting parallel computing model. Parallel platforms represent an efficient way to tackle sequences alignment problems. In parallel computing platforms, two or more processors can be using simultaneously for distributed workload, which represent a solution overwhelm a single sequential processor dilemma. Restricted memory space in sequence comparison problems is a challenging area in parallel computing. There is remain a need for an algorithm to harness additional processing power. The scaling of distributed memory enables considering larger sequences than any other possibility. Distributing data between multi-processors are a core concern in parallel platforms, where all processors apply the same workload on a different portion of the data. Resource sharing in parallel platforms improves performance parameters, while workload distribution, memory management, and communications represent curricular issues in order to minimize computing cost. In this paper, these obstacles studied with emphasis on the implementations of local sequence alignment using the SW algorithm.

Table 1. Time and space complexity for DP an HM in sequences alignment [6].

Author	Algorithm	Approach	Time Complexity
[1]	Global	Dynamic Programming	O(MN)
[2]	Local	Dynamic Programming	O(MN)
[3]	FASTA	Heuristic	O(MN)
[4]	BLAST	Heuristic	O(MN)

The rest of this paper organized as follows. Section 2 explained related work written on the scope of the subject. In the section 3, we briefly introduced local sequence alignment using SW algorithm, while, in section 4, the BWP model and scheduling technique are described in details. Results and discussion are presented in section 5. Section 6 concludes the paper with an outlook to future work.

2. RELATED WORK

The first serious discussions and analyses of long sequences emerged during space complexity in sequence comparisons. Memory constraint in long sequences is a prohibitive and compelling biologist to lose valuable information from newly discovered sequences. Few researchers have addressed the problem of space complexity in a long sequence. Linear space offers a mean of enhancing and improving space complexity in similarity detections for homologies sequences. [7], a pioneered in linear space implementation for sequences alignment problems proposed an exact algorithm to calculate global alignment between two sequences M and N in quadratic time. The proposed approach splits sequence M in the middle and generate subsequences M_1 and M_2, then calculate corresponding place for sequence N and finally, generate subsequences N_1 and N_2. This recursion roughly doubles execution time when compared with the original algorithm.[8], proposed z-align, a parallel strategy in limited memory space to reduce time and space needed in local alignments for comparing large sequences. However, the proposed strategy fails to compare more sequences length than any other known techniques. In an another attempts, many hardware accelerators are used to solve space complexity in long sequences such as Graphics Processing Units (GPU) [9-12], Field Programmable Gate Arrays (FPGA) [13-15], and Network-on-Chip (NoC) [16, 17]. Unfortunately, these methods do not always guarantee in comparing long sequences, because of memory limitations. Furthermore, high costs of these devices could hinder using such solutions; therefore, an alternative approach is necessary.

Most recent studies in sequence alignment problem have only been carried out using shared memory architecture [18-27]. However, a serious weakness with this architecture is the limitation and constraint of fixed sizes of memory available for all shared processors. This major drawback makes any algorithms and/or techniques for long sequences comparisons based on shared memory is unreasonable and impractical. Multicore platforms designed to work on shared memory architectures; a constraint for memory size would be a normal corollary in these platforms. Distributed Memory (DM) and Shared Distributed Memory (SDM) are prominent platforms tackle long sequences comparisons problems. A variety of algorithms is used to implement sequences alignment problems in DM and SDM [28-30]. Each has its advantages and drawbacks. Applicable parallel platform for DM and SDM uses multiprocessor's architecture, where each processor has it is own memory. However, one of the major drawbacks of multiprocessor platforms is communication complexity in shared processor.

One of the most current discussions in parallel computing is the task distribution between shared processors. However, difficulties arising when an attempt made to distribute tasks as well the system performance would increase significantly. To date various methods have been developed and introduced in workload and/or task distribution in sequence alignment problems. In most recent studies, workload distribution for parallelization of similarity the matrix in sequence alignment problems has been implemented in four different ways: substring of character [22, 24, 27, 28], rows and column [19, 23, 25], partitioning into segments [21, 26, 29, 30], and stages of distribution [18, 20, 31]. In substring of three or four characters, algorithms tend to distribute the workload between shared processors into groups of characters or substrings. Since sizes of these groups are too small compared to sequence length, thus a fine-grain parallelism is applied. Traditionally, fine-grain is very complex in communications; these can be time-consuming and are often technically difficult to perform. Row and column methods are distributing a workload as a matrix of rows and columns to every shared processor. Unfortunately, these methods do not always guarantee passing dependent cells for other processors. Furthermore, there are no clearly techniques adopting for controlling shared variables in memory. Partitioning into segments methods, always distribute workload dynamically into *k* parts with different sizes based on the machine identification number, program setting, and a number of register elements in the SIMD. However, normally dynamicity is a source of waiting time, which for long sequences is unreasonable and impractical. Finally, stages of distribution methods will assign workload in every stage according to periodic processing progress notifications or in super-master and producer model. Communications cost in these methods is always higher than others, which may cause delays in calculating results.

The previously mentioned studies reviewed so far, however, suffer from the fact that they ignoring or overlooking space complexity and scheduling technique between shared processors in order to accelerate long sequences comparisons.

3. LOCAL SEQUENCE ALIGNMENT USING THE SW ALGORITHM

Consider two sequences $A_i = (A_1,.., A_m)$ and $B_j = (B_1,..,B_n)$, local sequence alignment calculated using the following equation:

$$s(i,j) = max \begin{cases} s(i-1,j-1) + S(Ai,Bj) & \text{the first case} \\ s(i-1,j) - g & \text{the second case} \\ s(i,j-1) - g & \text{the third case} \\ 0 \quad \text{start an alignment from the begining} & \end{cases} \quad (1)$$

To obtain the scores $s(i,j)$, the partial alignment is divided into three cases occurred: (1) A_i matches to B_j, (2) A_i is alignment to gap, and (3) B_j is alignment to gap. At this stage, three values are evaluated i.e. $s(i\text{-}1,j)$, $s(i,j\text{-}1)$, $s(i\text{-}1,j\text{-}1)$. Furthermore, substitution matrix $S(A_i,B_j)$ must consider in order to predict biological relationship between two sequences, as well as a gap penalty *g*. In the first case the scores $s(i,j)$ is the sum of the score for the alignment of the substring $(A_1,..,A_{i-1})$ and the substring $(B_1,..,B_{j-1})$ in accordance with the substitution matrix $S(A_i,B_j)$. In the second case the gap open penalty is deducted from the score of the alignment of substrings $(A_1,..,A_{i-1})$ and $(B_1,..,B_{j-1})$. The third case is analogous to the second case. Finally, a zero case ignores negative alignment score in recursion way.

4. BLOCK WISE PARADIGM FOR SPACE COMPLEXITY IN SEQUENCE ALIGNMENT USING SCHEDULING TECHNIQUE FOR COMMUNICATION

Usually SW costs $O(mn)$ space and times, which for long sequence's length is prohibitive and impractical. A key aspect for parallelizing SW is time and space complexity, which will reduce through distributing workload to multi-processors, that able to calculate small portion of data with other shared processors. Usually, data dependency in filling matrix stage is achieved using *Wavefront* method [32]. However, this method leads to flood the communications between shared processors. An acceptable way is to distribute the matrix *S* in blocks. Blocks method is suitable for Single Program Multiple Data (SPMD) architecture where every processor has its own memory. In this architecture, every shared processor executes the same program and has its own private data, $L = (L_0, ..., L_n)$. Furthermore, every processor produces an output $U = (U_0, ..., U_n)$. In this paper an extension of Bulk Synchronous Parallelism (BSP) model [33] called Block Wise Paradigm (BWP) is adopted, which represent a coarse grain parallelism. In BWP, each processor accomplishes selected number of blocks. It is mainly proposed by [34] for string matching and editing with target to minimize the complexity of searching for matching in parallel platform. However, with some amendment of adding scoring scheme, the model can also be used on sequence similarity detection. Coarser granularities provide less communication between processor than finer granularity [35]. In every block, a number of consecutive rows are attached to every processor. The number of rows depends on the query sequence size. The first row R_{11} in the first block K_1 is computed by the first process P_1, then the first row R_{12} in the second block K_2 is computed by the second process P_2, while the first process P_1 calculate the second row R_{21} in the first block K_1 and so on. After each process finish, one row R_{1n} a scheduling technique is adopted to send the result to the next process P_{i+1} in the next block K_{i+1} $(0=<i=<n)$. The process P_{i+1} starts after the process P_i sends the last row of the block, which urgently needed for calculating the block of process P_{i+1}. Figure 1 depicts a comparison of two sequences using similarity matrix S, to allocate an optimal result. It shows the distribution of blocks for every processor, where a column of similar color is assigned to one process for computation. When each processor P_i is finished every row R_i at every time, it moved to the next row R_{i+1} and sent the last element *Ri* to the next process *Pi+1*. The elements in yellow color with a graphic arrow are the ones sent by the previous process P_i to the next process P_{i+1}. The first process P_0 does not receive any element from any process and the last process P_3 do not send any element to any process.

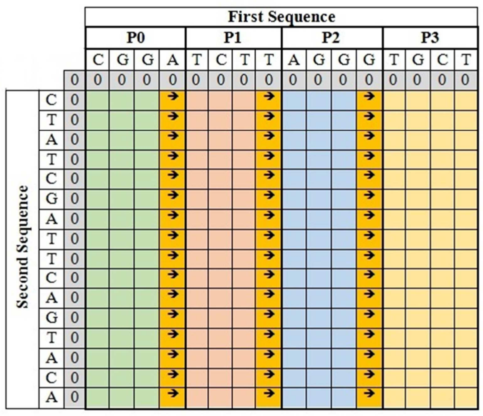

First Sequence
P0
P1
P2
P3
C G G A T C T T A G G G T G C T
Second Sequence
C T A T C G A T T C A G T A C A
0

Figure 1. Blocks of sub-sequences in BWP

BWP is applicable to be executed in scheduler-worker architecture, using cluster of computers, where the scheduler shares in the matrix calculations and has the responsibility for controlling shared workers. In the next section, fully details on using this architecture are explained.

4.1 A cluster architecture for similarity detection between two sequences using BWP

The dominant of High Performance Computing (HPC) architectures currently and for the foreseeable future comprise clusters of nodes interconnected through a high-speed network [36]. The two key reasons for using clusters are performance and scalability. Cluster consists of many different hardware and software components with complex interactions between various components. This architecture is suitable for sequence alignment problem, where each shared processor performs independent task and/or job. The main process farm employed in this paper is using one processor as a scheduler or a manager, while other processors act as workers. The Scheduler-Workers model is a widely used in parallel computing to implement dynamic programming algorithm. The model is ideally suited to parallel computation of large numbers of identical and independent tasks each of which constitutes a relatively small fraction of the total work. The scheduler processor reads a query and reference sequences with some other parameters and divides the reference sequence between workers into P blocks, where P is the number of the shared processors in the model. After performing SW algorithm, all workers send the results to the scheduler.

4.1.1 Scheduler side algorithm

On scheduler side algorithm, one processor P_0 acts as a scheduler managing and sending a query sequence and the blocks K_i of a reference sequence to every worker P_i, which in turn, perform Pairwise Local Sequence Alignment (PLSA) using an SW algorithm with the query sequence. The distribution of the blocks K_i of a reference sequence is invoked by the scheduler. When the Scheduler finishes alignment for one row in assigned block, it sends the last cell to the neighbor worker. This process will be continued until all rows in the block are processed. Finally, the scheduler outputs the optimal result of an SW algorithm after gathering all blocks from all workers. The following pseudo code for describing gathering blocks from all workers.

Algorithm 1: **/* Gathering results from workers by scheduler */**

```
Gather_From_Workers()
1:  Begin
2:  Reserve_Memory_Allocating_Every_Worker_
    Matrix;
3:  For (Worker_Rank = 1; Worker_Rank <
    Total_Workers; Worker_Rank ++)
4:  {
5:  Scheduler_Receive(Worker_Block_Matrix);
6:  }
7:  For (k = 0; k < Sequence_A; k++)
8:  For (l = 0; l < Sequence_B; l++)
9:  {
10  Fill_ Rows and Column_in_Scheduler_Matrix;
11  }
12  End
```

Journal of Theoretical and Applied Information Technology
20th May 2014. Vol. 63 No.2
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

In accordance with the set theory, for every A_i and B_j, is a subset from C_k, where $A_i \neq B_j$, then $A_i \cup B_j$ has also been a subset from C_k

$$C_k = A_i \cup B_j \tag{2}$$

Corollary: $f : S \rightarrow \{1, \ldots, n\}$ and $S = \{X_1, X_2, \ldots, X_n\}$ then

$$S = X_1 \cup X_2 \ldots, \cup X_n \tag{3}$$

(2) and (3) conclude this observation:

Observation: For every two sequences $A_i = (A_1,.., A_m)$ and $B_j = (B_1,..,B_n)$, Suppose $C_\circ = (C_{\circ 1},.., C_{\circ u})$, and $C_\Delta = (C_{\Delta 1},.., C_{\Delta z})$, are aligned for A_i and B_j, then

$$C_\circ \cup C_\Delta \text{ is also an alignment for } A_i \text{ and } B_j \tag{4}$$

The observation and proof (4) imply that in scheduler-worker model if any worker sends the optimal alignment to the scheduler then the final output from scheduler after gathering of all workers will be an optimal alignment for the comparison between any two sequences. The following pseudo code describes the scheduler side algorithm.

Algorithm 2: **/*Scheduler Side Algorithm*/**

```
1:  Begin
2:  Declare arguments of the program;
3:  Reserve_Memory_for_SequenceA =
        SequenceA_Length+1;
4:  Reserve_Memory_for_SequenceB =
        SequenceB_Length+1;
5:  Initiate_Array H[A][B] = 0;
6:  Broadcast(&Gap, &Match/Mis,
        Sequences_Size_A_B);
7:  Distribute (Blocks_A, Blocks_Size,
        Displacement, A_partition);
8:  Scheduler_Size = Get_My_Columns(MyId,
        Total_Workers, N_b + 1);
9:  Smith-Waterman_Algorithm();
10: Send_Block_Cell_To_Worker_Neighbor
        (&H[Row][MyBlock], Rank+1);
11: Get Algorithm 1: Gather_From_Workers();
12: Smith-Waterman_Algorithm();
13: for(i=0;i<N_b;i++)
        for(j=0;j<N_a;j++)
        Print (Smith-Waterman_Result);
14: End
```

4.1.2 Worker side algorithm

On the other hand, every worker receives from the scheduler a query sequence and one block of the reference sequence. Neighbor worker of the scheduler receives the last cell and performs SW algorithm. When every worker finishes in filling one row of a assigned block, it sends the last cell to the neighbor. This process will continue until all rows in the block are processed. The results generated by workers is sent to the scheduler. The following pseudo code describes the worker side algorithm.

Algorithm 3: **/*Workers Side Algorithm*/**

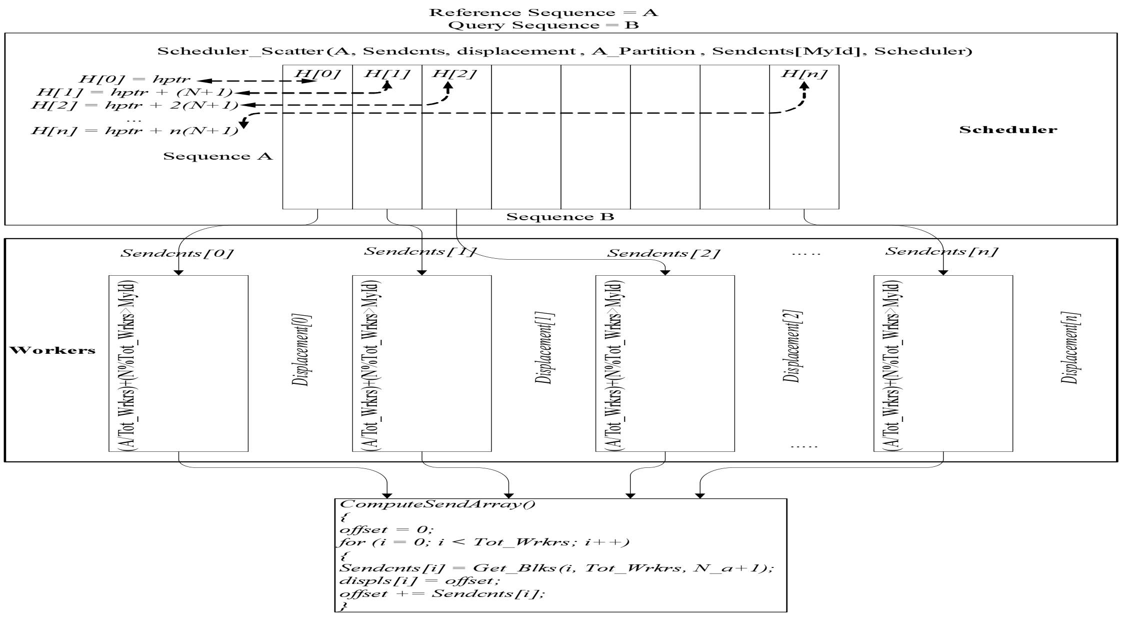
```
1:  Begin
2:  Declare variables;
3:  My_Columns = GetNoColumns(MyId,
        Total_Workers, N_a + 1);
4:  Block (N_a+1) memory for sequence A;
5:  Block (N_b+1) memory for sequence B;
6:  Initiate_Array H[Rows][Columns] = 0;
7:  for (i = 1; i < Rows; i++)
8:  {
9:  Current_Worker_Receive(&Last_Cell_Value,
        MyId-1);
10: Smith-Waterman_Algorithm();
11: for (j = 1; j < Columns; j++)
12: {
13: Smith-Waterman_Algorithm();
14: } // For_Columns_End
15: Send_Block_Cell_To_Worker_Neighbor
        (&Max_Value, MyId + 1);
16: } // For_Rows_End
17: Send_Matrix_To_Sceduler
        (H[Rows][Columns], Scheduler);
18: End
```

4.2 Space complexity in BWP

Memory architecture becomes a key issue in parallel platform, which it frequently determines the optimal programming model. In BWP, any processor has it is own local memory; addresses in one processor do not map to another processor. Blocking is an efficient way to exploit local memory and fine grain parallelism [37]. Figure 2 depicts a real image from program execution and describes the overall memory consumption in Scheduler-Worker model in both sides when scheduler scattering blocks to every worker and sizes of blocks in every worker. However, estimation of memory consumptions through analysis improves results extracts and determines the weaknesses in the model.

Proposition: Given W_j workers and partitions of aligned sequences Q_i from A_i with $|Q_i| \leq W_j$, an optimal alignment between A_i and B_j is found in $O(\frac{m+\frac{n}{w}}{w})$ space, where m length of query sequence B_j, n length of reference sequence A_i, and w are the number of shared workers.

Proof: Suppose W_j are workers $W_1, W_2, \ldots, W_j$, to solve the sequence alignment problem for two sequences $A_i = (A_1,.., A_m)$ and $B_j = (B_1,..,B_n)$, also assume that $A`$ and $B`$ partitions of A_i and B_j $(A` < A_i \; and \; B` = B_j)$ is distributed across workers, then each worker hold $A`$+$B`$ data. Using equation 1 with respect to space complexity for SW algorithm shown in Table 1, an alignment of partitions $A`$ and $B`$ is achieved in the sequential machine on *O(rs)* space $(s = n \; and \; r < m)$, where r and s are sequence's length of $A`$ and $B`$ respectively. Using parallel technique discussed in section 4, every worker thus executes SW in $O(s + \frac{r}{p})$. Using equation 2, 3, and 4 are then an optimal alignment of SW achieves in in $O(\frac{m+\frac{n}{w}}{w})$ space.



```
ComputeSendArray()
{
offset = 0;
for (i = 0; i < Tot_Wrkrs; i++)
{
Sendcnts[i] = Get_Blks(i, Tot_Wrkrs, N_a+1);
displs[i] = offset;
offset += Sendcnts[i];
}
```

Figure 2. Scheduler Scatter Evenly Blocks To Every Worker.

4.3 Communications costs in BWP

In large data system, the cost of communications is very high due to the tremendous data travelling through nodes in any requests and/or data exchanges. Time delays occur in data communication between processors represents a crucial concern for parallel computation efficiency. One of the solutions for minimizing communications between nodes in parallel platforms is to adopt a coarse-grain parallelism. Block Wise Paradigm (BWP), the model adopted in this paper for distributing reference and query sequences between workers sustain in the reduction of communication between workers substantially. Communication between workers and scheduler occurs only in two cases, when the scheduler sends a query sequence and block of the reference sequence, and when the workers send results back to the scheduler. Another communication takes place between shared workers when every worker sends the last cell to the neighbor, in this case the communication is iterated until all rows in every block finish. This section analyzes the communication complexity of BWP including synchronizations overhead.

The Block Wise Paradigm (BWP) abstracts the communications operations in a sequential synthesis of global Supersteps [33, 34]. These Supersteps conceptually occupy the full width of the executing architecture, it includes the following operations:

- Local computation.
- Communications.
- Synchronization barrier.

Abstraction of operations in Supersteps appears in the following pseudo code:

Superstep1: Local computation phase *[n/w]*. For all worker W_i hold a local partition *A[i]* of A_i, *m=Max (A[i])*, where *m* is the last cell in each block.

Superstep2: Communication phase *[gh, with h <* A_i*]*, where *h* defined sends and receives at *h* data letters, $h = max\{h_s, h_r\}$, *g* measures ability of the network to continuous traffic, and *gh* is the communication time per data letters.

```
If my PID !=0 send (m)
Else
for each i in {1… p-1} recv (m, i)
If my PID=0 for each i in {1… p-1} m=max (m, $m_i$)
```

Journal of Theoretical and Applied Information Technology
20th May 2014. Vol. 63 No.2
© 2005 - 2014 JATIT & LLS. All rights reserved·
ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

Superstep3: Synchronization barrier l, identifying waiting state for workers until reach barrier. Barrier is due to fixed overhead, such as start-up costs of sending data and costs of checking whether all data arrived at their destination.
Superstep2 and 3 lead to calculating the communication cost which includes synchronization. Thus, the total cost is defined by the following equation:

$$T(h) = hg + l \quad (5)$$

Communication cost of BWP in an abbreviate formula is an expression of the following

$$T = W + L \quad (6)$$

W is the maximum number of Floating-point Operations per Second (FLOPS) of the processor in the Superstep. To measure T, a wall clock needed to give the elapsed time. Because barriers make circularities in data dependency impossible, L value must near to zero [38].

5. RESULTS AND DISCUSSION

The main intention of this section is to evaluate and test BWP model and communication complexity as well as synchronization. Analyzing parallel models requires evaluating the performance of resources involved such as the number of shared processors, space, and communication. This section measures space and communication complexity of BWP using scheduler-worker paradigm. The experiments are conducted on a dedicated X86-based PC with Intel(R) Core(TM) i7-2670QM CPU 2.20 GHz, 2201 MHz, 4 Core(s) 8 Logical Processor(s). Installed physical memory (RAM) is 8.00 GB, running over MS Windows 7 Service Pack 1. The proposed algorithm is implemented using C++ while the parallel version is executed using Message Passing Interface (MPI). The standard library is based on unanimity of the MPI forum to establish portable and efficient standard for writing message-passing programs. MPICH2, a high-performance and portable implementation of the MPI is used to manage communications between shared processors. Real datasets of DNA and Protein sequences are obtained from National Center for Biotechnology Information [39] using CLC Sequence Viewer, a GUI bioinformatics software environment. For comparing the results, different sizes of twenty sequences are obtained in the range of 411 KBP to 4 MBP.

5.1 Estimation of space complexity in BWP model

An optimal alignment between two sequences can be found in $O(\frac{m+\frac{n}{w}}{w})$ space, where m and n are the lengths of the query and reference sequences, respectively, w is the number of shared workers. To measure memory consumption in BWP model, two sequences with length 411120 BP and 418695 BP of nucleotides are calculated using cluster of 8 workers.

In Figure 3, it is clearly shown that memory sizes dropdown during the numbers of workers increase, which is normally due to decreasing of block sizes in BWP when using multiple workers. However, as an evident result when using one worker the amount of memory needed is 830 K, while in case of two workers only 310 K is needed; this surprising results, in fact, shows the complication of executing SW in serial machines, which is more than the tendency of implementing SW in parallel machines. Another observation form Figure 3 is that once applying two or more workers in implementing the SW algorithm using BWP model the complexity in memory is closely related such that using 3, 4, 5, 6, 7, and 8 workers imply 184, 129, 99, 80, 67, and 58 K respectively. An important relation displayed in Figure 4 as an exponential growth when reducing the numbers of workers memories sizes increase conversely.

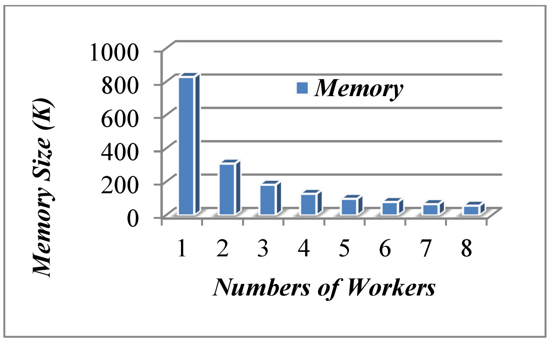


Figure 3. Amount of memory sizes involves in BWP using 8 workers. The results observed and recorded when calculating 400×400 KBP nucleotides.

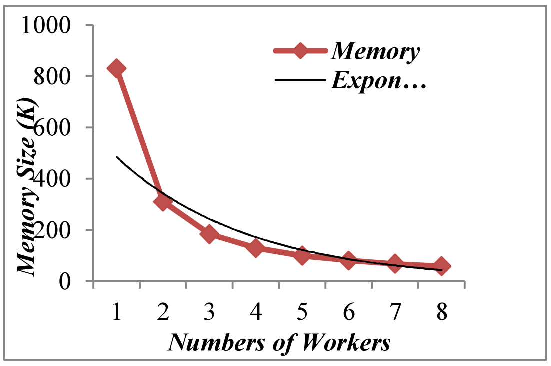


Figure 4. Exponential Forecasting Of Memory Consumption Using 8 Workers

5.2 Estimation of communications costs using BWP model

BWP consists of a sequence of supersteps. In each superstep, every worker performs local computations, initiates communication to another worker, and synchronizes it at the end. Running workers defined earlier remain constant during the execution time. Average time T of the proposed algorithm runs on $\frac{T}{W}$ time, where W is the number of shared workers. However, the cost of communications determined by equation (5) is as follows:

$T(h) = hg + l$

Where, h defined max sends and receives operations on data, the g measured ability of the network to deliver data, l is the cost of a barrier synchronization of data, and hg is the communication time per data letters.

In order to predict BWP performance, values for parameters h, g and l must be identified. However, estimating values for these parameters relies on different dependencies based on the parallel architecture which used for running algorithm. In cluster architecture, length of a message travelled through shared workers sometimes effecting the performance of an overall system. However, the BWP model is an extension of BSP model, where there is no distinction between different messages lengths [38]. Oxford BSPLIB [40], is a parallel programming library substitutional to MPI and PVM. It offers a toolset including profiling tools and implementations of the library for many different machines and focuses on providing easiest portable implementations with accurate predictions of performance parameters using the BSP model. BSP machine parameters g and l are calculated on BSP cluster similar to the proposed experiment architecture in this paper. BSP cluster includes eight 400Mhz Pentium IIs with 128Mbytes of memory connected by a 100Mbps Ethernet switch. Worst-case results are obtained by using g and l parameters from BSPLIB toolset with known sequence length. Therefore, output results measure a highest communications cost, which is then used as a benchmark for evaluating the performance of the proposed algorithm in this paper. The values of the g and l parameters listed in the Table 2 with a total communication cost $T(h)$, while illustrating graph appears in Figure 5. For more accurate measurement parameters are calculated in **F**loating-point **O**perations **P**er **S**econd (**FLOPS**).

Table 2. Values For Parameters L (Flops) And G (Flops/Word) As Estimated By Oxford BSP Toolset [40]. However T(H) Calculated By Equation 5.4 Using Tested Sequence With Length 100 BP Residues

w (No. of workers)	l (flops)	g (flops/word)	T(h)Second
1	128	1.3	4.30
2	5654	33.5	150.07
4	11759	31.5	248.48
8	18347	30.9	357.28

As can be seen in Figure 5, the implementation of the proposed algorithm is measured by 100 BP nucleotides over 8 workers which increases the cost of communication as the number of workers increases. As a normal corollary in parallel computing execution times decrease, during the cost of communication increase in large data system [37, 41]. However, the proposed algorithm in this paper still keeps competitive results in a cost of communications bottleneck. Using one worker cause 4.30 communications complexities, which is very less than using other workers; this cause by less communication required by the proposed algorithm. Linear relation between parameters appears clearly in Figure (5); also linear growth forecasting continues ascending indirectly proportional between the cost of the communications and the number of workers.

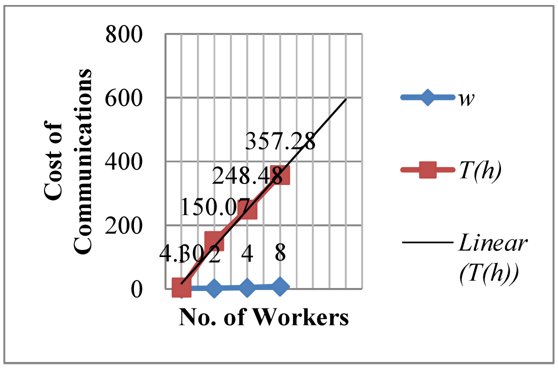


Figure 5. The Communication Cost Of 8 Workers With Linear Growth Forecasting

6. CONCLUSION

This paper has gone some way towards enhancing our understanding of DNA and protein conformations by considering a parallel model to detect similar regions in the comparisons of long sequences with some applicable algorithms. One of the more significant findings to emerge from this paper is the evolutionary relationship that can be detected between compared sequences, as well as similar proprieties and structure. This work contributes to the existing knowledge by providing a complete solution to implement SW algorithm in a parallel architecture using Block Wise Paradigm (BWP). Furthermore, this paper has given an account of the reasons for the widespread use of Scheduler-Worker model as well as cluster of workers. Finally, mathematically and statically analysis of space complexity and communications costs in BWP is done in order to prove competitor results. More research needed to better understands the impact of adopting load-balancing techniques on the communication costs and/or space complexity. Future research should, therefore, concentrate on applicable parallel platforms for DM and SDM with multiprocessor's architecture.

REFRENCES:

[1] S. B. Needleman and C. D. Wunsch, "A general method applicable to the search for similarities in the amino acid sequence of two proteins," *Journal of molecular biology,* vol. 48, pp. 443-453, 1970.

[2] T. Smith and M. Waterman, "Identification of common molecular subsequences," *J. Mol. Bwl,* vol. 147, pp. 195-197, 1981.

[3] W. R. Pearson and D. J. Lipman, "Improved tools for biological sequence comparison," *Proceedings of the National Academy of Sciences,* vol. 85, p. 2444, 1988.

[4] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman, "Basic local alignment search tool," *Journal of molecular biology,* vol. 215, pp. 403-410, 1990.

[5] W. Haque, A. Aravind, and B. Reddy, "Pairwise sequence alignment algorithms: a survey," in *Proceedings of the 2009 conference on Information Science, Technology and Applications*, 2009, pp. 96-103.

[6] L. Hasan, Z. Al-Ars, and S. Vassiliadis, "Hardware acceleration of sequence alignment algorithms-an overview," 2007, pp. 92-97.

[7] D. S. Hirschberg, "A linear space algorithm for computing maximal common subsequences," *Communications of the ACM,* vol. 18, pp. 341-343, 1975.

[8] R. B. Batista, A. Boukerche, and A. C. M. A. de Melo, "A parallel strategy for biological sequence alignment in restricted memory space," *Journal of Parallel and Distributed Computing,* vol. 68, pp. 548-561, 2008.

[9] M. Schatz, C. Trapnell, A. Delcher, and A. Varshney, "High-throughput sequence alignment using Graphics Processing Units," *BMC bioinformatics,* vol. 8, p. 474, 2007.

[10] S. Manavski and G. Valle, "CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment," *BMC bioinformatics,* vol. 9, p. S10, 2008.

[11] Y. Zhang, S. Misra, D. Honbo, A. Agrawal, W. Liao, and A. Choudhary, "Efficient pairwise statistical significance estimation for local sequence alignment using GPU," 2011, pp. 226-231.

[12] P. Borovska and M. Lazarova, "Parallel models for sequence alignment on CPU and GPU," 2011, pp. 210-215.

[13] X. Meng and V. Chaudhary, "Boosting data throughput for sequence database similarity searches on FPGAs using an adaptive buffering scheme," *Parallel Computing,* vol. 35, pp. 1-11, Jan 2009.

[14] J. Allred, J. Coyne, W. Lynch, V. Natoli, J. Grecco, and J. Morrissette, "Smith-Waterman implementation on a FSB-FPGA module using the Intel Accelerator Abstraction Layer," in *Parallel & Distributed Processing, 2009. IPDPS 2009. IEEE International Symposium on*, 2009, pp. 1-4.

[15] C. YILMAZ and M. GÖK, "System designs to perform bioinformatics sequence alignment," *Turkish Journal of Electrical Engineering & Computer Sciences,* vol. 21, pp. 246-262, 2013.

[16] S. Sarkar, G. R. Kulkarni, P. P. Pande, and A. Kalyanaraman, "Network-on-chip hardware accelerators for biological sequence alignment," *Computers, IEEE Transactions on,* vol. 59, pp. 29-41, 2010.

[17] D. Díaz, F. J. Esteban, P. Hernández, J. A. Caballero, G. Dorado, and S. Gálvez, "Parallelizing and optimizing a bioinformatics pairwise sequence alignment algorithm for many-core architecture," *Parallel Computing,* 2011.

[18] F. M. Mendonca and A. C. M. A. d. Melo, "Biological Sequence Comparison on Hybrid Platforms with Dynamic Workload Adjustment," in *Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International*, 2013, pp. 501-509.

[19] N. Alachiotis, S. Berger, T. Flouri, S. P. Pissis, and A. Stamatakis, "libgapmis: extending short-read alignments," *BMC Bioinformatics,* vol. 14, p. S4, 2013.

[20] H. Martínez Pérez, J. Tárraga, I. Medina, S. Barrachina, M. I. Castillo Catalán, J. Dopazo, *et al.*, "Concurrent and Accurate RNA Sequencing on Multicore Platforms," 2013.

[21] N. Neves, N. Sebastiao, A. Patricio, D. Matos, P. Tomás, P. Flores, *et al.*, "BioBlaze: Multi-core SIMD ASIP for DNA sequence alignment," in *Application-Specific Systems, Architectures and Processors (ASAP), 2013 IEEE 24th International Conference on*, 2013, pp. 241-244.

[22] D. Satyanvesh, K. Balleda, and P. Baruah, "Genalign—A high performance implementation for aligning the compressed DNA sequences," in *Advanced Computing Technologies (ICACT), 2013 15th International Conference on*, 2013, pp. 1-6.

[23] N. Sebastião, G. Encarnação, and N. Roma, "Implementation and performance analysis of efficient index structures for DNA search algorithms in parallel platforms," *Concurrency and Computation: Practice and Experience,* 2012.

[24] D. Satyanvesh, K. Balleda, A. Padyana, and P. Baruah, "GenCodex-A Novel Algorithm for Compressing DNA sequences on Multi-cores and GPUs," in *19th IEEE International conference on High Performance Computing., December 2012.*, 2012.

[25] G. Delgado and C. Aporntewan, "Data dependency reduction in Dynamic Programming matrix," in *Computer Science and Software Engineering (JCSSE), 2011 Eighth International Joint Conference on*, 2011, pp. 234-236.

[26] P. Borovska, V. Gancheva, G. Dimitrov, and K. Chintov, "Parallel performance evaluation of multithreaded local sequence alignment," 2011, pp. 247-252.

[27] S. Bandyopadhyay and R. Mitra, "A parallel pairwise local sequence alignment algorithm," *NanoBioscience, IEEE Transactions on,* vol. 8, pp. 139-146, 2009.

[28] A. Montanola, C. Roig, and P. Hernandez, "Pairwise sequence alignment method for distributed shared memory systems," in *Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on*, 2013, pp. 432-436.

[29] A. Nordin, M. Yazid, A. Aziz, and M. Osman, "Parallel Guided Dynamic Programming Approach for DNA Sequence Similarity Search," 2009.

[30] M. Nordin and A. Rahman, "Utilizing MPJ Express Software in Parallel DNA Sequence Alignment," 2009, pp. 567-571.

[31] C. Wu, A. Kalyanaraman, and W. R. Cannon, "A scalable parallel algorithm for large-scale protein sequence homology detection," in *Parallel Processing (ICPP), 2010 39th International Conference on*, 2010, pp. 333-342.

[32] A. Wozniak, "Using video-oriented instructions to speed up sequence comparison," *Computer applications in the biosciences: CABIOS,* vol. 13, pp. 145-150, 1997.

[33] L. G. Valiant, "A Bridging Model for Parallel Computation," *Communications of the Acm,* vol. 33, pp. 103-111, Aug 1990.

[34] C. E. Alves, E. N. Cáceres, and F. Dehne, "Parallel dynamic programming for solving the string editing problem on a CGM/BSP," in *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures*, 2002, pp. 275-281.

[35] K. Hamidouche, F. M. Mendonca, J. Falcou, A. C. M. A. de Melo, and D. Etiemble, "Parallel Smith-Waterman Comparison on Multicore and Manycore Computing Platforms with BSP++," *International Journal of Parallel Programming,* pp. 1-26, 2012.

[36] H. Q. Jin, D. Jespersen, P. Mehrotra, R. Biswas, L. Huang, and B. Chapman, "High performance computing using MPI and OpenMP on multi-core parallel systems," *Parallel Computing,* vol. 37, pp. 562-575, Sep 2011.

[37] F. Yu and C. Coarfa, "Sequence Alignment, Analysis, and Bioinformatic Pipelines," in *Next Generation Sequencing*, ed: Springer, 2013, pp. 59-77.

[38] D. B. Skillicorn, J. M. Hill, and W. F. McColl, "Questions and answers about BSP," *Scientific Programming,* vol. 6, pp. 249-274, 1997.

[39] NCBI. (2014, 15 Feb). *National Center for Biotechnology Information.* Available: http://www.ncbi.nlm.nih.gov/

[40] Oxford BSPLIB. (2014, 25 Feb). *Oxford Parallel.* Available: http://www.bsp-worldwide.org/implmnts/oxtool/

[41] J. T. Dudley and A. J. Butte, "A quick guide for developing effective bioinformatics programming skills," *PLoS Comput Biol,* vol. 5, p. e1000589, Dec 2009.