Indian Journal of Science and Technology, *Vol 7(5), 589–599, May 2014*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Coarse Grain Load Balance Algorithm for Detecting Similar Regions in DNA and Proteins Sequences

Manhal Elfadil Eltayeeb Elnour*, Muhammad Shafie Abd Latif and Ismail Fauzi Isnin

Department of Computer Science, Faculty of Computing, Universiti Teknologi Malaysia, Johor, Malaysia; manhalus@gmail.com, shafie@utm.my, ismailfauzi@utm.my

Abstract

The tremendous quantity and quality of data obtained by conformations of DNA and protein sequences makes their analysis very time consuming, complex, expensive and impractical. Therefore, the feasible way to identify new sequences is to compare them with well-known sequences available in established genetic databank. Comparing sequences may reveal functional, structural, and evolutionary analogies between sequences. Needleman and Wunsch (NW), and Smith and Waterman (SW) algorithms are pioneers in dynamic programming matrix for comparing two sequences with gap penalty function. However, for long sequences both methods contribute toward time and space complexity. FASTA and BLAST are heuristic method based on hitting techniques for fast detection of similar region; unfortunately they produced results with no sensitivity. There remains a need for an efficient method(s) that can detect similar regions in two sequences with accurate results and reasonable time. In this paper, we extend an existing approach to develop an efficient parallel algorithm for pairwise local sequence alignment. Our method is based on load-balancing algorithm with the CPU scheduling technique in order to accelerate the calculation of data-dependency problem in sequences alignment. Using X86-based PC with eight logical processors we able to apply 4 MBP on the proposed algorithm with a speedup of 33% increase compared to the original SW algorithm.

Keywords: DNA, Parallel Computing, Protein, Sequence Alignment, Smith-Waterman

1. Introduction

The easiest way to identify new sequences is to compare them with a well-known sequence, established and sorted in popular and generic data-banks. Usually, a biological sequence involves comparing against hundreds and/or thousands of sequences. The most important gene repositories include GenBank, NCBI, EMBL, DDBJ, EMDB, and PDBe. These organizations exchange data daily and continuously generate a new release of sequences. Comparing two sequences may reveal or predict functional, structural, and evolutionary analogies of the compared sequences. Sequences comparisons, sequences alignment, and similarity detection, are the same acronyms for finding an approximate pattern matching between biological sequences[1]. Alignment of two or more sequences shows similar and different regions between sequences. The correspondence between similar subsequences represents important information for biologists. Alignments encompass pair-wise alignment (particular for two sequences) and multiple alignments dedicated in aligning a number of sequences[2]. Most two approaches for aligning pair-wise sequences are global and local. Global alignment is convenient if sequences are compared as a whole, and/or compared sequences are homologous across their entire length[3]. Local alignments are appropriate for detecting specific conserved regions, and/or to obtain similarity between parts of sequences.

Most of alignments algorithms are based on Dynamic Programming (DP) or Heuristic Methods (HM). DP algorithms guarantee the best possible alignment between two sequences (optimal alignment), while heuristic algorithms guarantee fast solutions for approximate optimal results[4–7]. Needleman and Wunsch[8] were the pioneer in using DP to search for global optimal alignment and it was extended to local sequence alignment by Smith and Waterman[9]. Using DP methods in the sequences alignment will consider accurate results with optimal

*Author for correspondence

alignment. However, for long sequence's length and in searching entire Bio-databases these methods tend to be very slow. Heuristic algorithms such as FASTA[10] and BLAST[11] were developed to speed up sequences similarity detection in Bio-databases while attempting to keep sensitivity as higher as possible. Both FASTA and BLAST are much faster, but do not guarantee optimal and accurate results[12]. This paper focuses on implementing local sequence alignment for pair-wise sequences to obtain accurate results with reasonable time.

While sequential machine architectures strive to show increasing in performance, the volume of bio-data is extremely increasing in tremendous scene. Furthermore, sequences alignment algorithms can achieve much better parallelization, by deconstructing sequences into a large and/or small number of independent tasks with a little bit of message passing communications between workloads. The data is passed to the appropriate processors properly and the final results are then assembled as the independent workloads complete.

The rest of this paper is organized as follows: Section 2 explains related work written on the scope of the subject. In section 3, we briefly introduce local sequence alignment using SW algorithm, while, in section 4, the proposed algorithm with 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

Various hardware accelerators were used for detecting similarity in DNA or proteins sequences[12] such as Graphics Processing Units (GPU)13–16, Field Programmable Gate Array (FPGA)17–19, and Network-on-Chip (NoC)[3,20]. Unfortunately, these methods do not always satisfying in comparing long sequences, due to the memory restriction. Furthermore, high costs of these devices may hinder in using such solutions and therefore, an alternative approach is necessary.

Striking feature in a powerful method is to utilize a profitable technology to play a major part in aligning long sequence length with no costs. Parallel computing is a promising method attracted much attention to tackle sequence alignment problems. Implementing sequence alignment in parallel platforms requires a parallel algorithm running on multiprocessor or multicomputer systems to accelerate the comparing process[21]. Furthermore, a suitable algorithm to distribute of workload between shared processor is a key technique more than sequences alignments process itself. In this paper, we aim to increase efficiency in comparing sequences by minimizing interactions between parallel processes and reducing the number of task divisions. Many researches in recent years have focus on accelerated sequences alignment problems using parallel platforms. However, the quadratic space complexity remains a problem in large sequences comparisons[5]. Furthermore, there is lack of studying and/or evaluating parallel techniques such as load balancing techniques, CPU complications in processing waiting-queue, and communication delay for messages between shared processors in parallel architecture. Therefore, there is in-need for rigors solution to harness a huge power obtainable from parallel platforms.

The SW algorithm finds similar regions between two sequences using dynamic programming techniques. The algorithm compares two sequences locally on a character-to-character level, and detects subsequences that were potentially similar[22]. Optimal local alignments could be defined by comparing query sequence with the reference sequence defined in Bio-databases. Most of a current discussions in local sequence alignment are focused on multicore architectures with shared memory (see Figure 1) including divide and conquer techniques[23–25], striped SW[26–28,68], Instruction-set Processor (ASIP) architecture[29], data compression[30,31], genome assembly (re-sequence) algorithms[32,33], and Symmetric Multi-Processing (SMP) architecture[34,35].

Generally, for evaluation, Luecke[36] listed difficulties of using shared memory architectures including lack of parallel programming models, lack of standards, and immaturity of multicore specific development and debug software tools. These obstacles compel researchers to focus on implementing SW in distributed memory where each processor has its own resources. A central problem

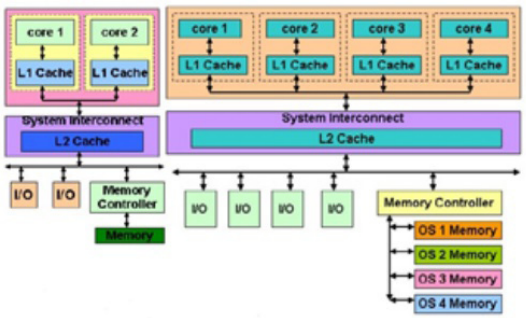

Figure 1. Dual core and Quad core Multicores processors.

of implementing the SW algorithm in multicore and/or SMP architectures is shared memory. In short sequence length this architecture be able to obtain possible results. However, deficiency arises for long sequence length, which is definitely unaccepted and prohibitive.

Distributed Memory (DM) systems normally consist of multiple-processor machines where each processor has its own memory. Computational jobs are operated on local data for each processor, while remote data requires complex communications with other processors. An advantage of using DM systems includes feasibility of increasing further numbers of processors as well as memory in order to increase system throughput and efficiency. Furthermore, each processor works with it is own data within local memory. However, the major drawback in DM system includes adopting efficient load balancing algorithm in order to minimize inter-processor communications[37].

Developing parallel algorithm for sequences alignment is daunting work and required in-depth knowledge of bioinformatics discipline as well as parallel techniques[38,39]. Few researchers have addressed implementing SW algorithm on a distributed memory, their contributions include divide and conquer algorithms[7,40–43], and the wave-front methods[4,45].

Generally, for evaluating Distributed Memory (DM) and Shared Distributed Memory (SDM) Sánchez et al.[45,46] analysed the performance of the search and a parallel application was implemented using SW algorithm on shared and distributed memory architecture machines. This study suggests an efficient use of coarse and fine grain parallelism to accelerate the sequence comparisons. However, the limitation of memory could be prohibitive. An observation in the study concludes the possibility to minimize the impact of memory latency by adopting double buffering techniques with large data blocks. Furthermore, a strategy is based on preventing a worker to wait for the response of a previous signal. This can minimize synchronization overhead and maximize the performance of the application.

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 using the SW algorithm is calculated in three parts as follows:

Part 1: Initialization: A and B are two sequences with the length m and n, respectively. A_i denotes *i-th* letter in a sequence A, and B_j denotes *j-th* letter in a sequence $B\left(1 \le i \le n, 1 \le j \le m\right)$. For aligning both sequences A and B, we introduce the matrix $s(i, j)$, which the elements represent scores for an alignment of *i-length* prefix of the sequence A, and *j-length* prefix of sequence B. Optimal alignment is achieved by computing $s(n, m)$ through an evaluation of an intermediate alignment for *i-long* prefix of A and *j-long* prefix of B.

Part 2: Filling matrix: To consider the scores $s(i, j)$, the partial alignment is divided into three cases: (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 $s(i - 1, j)$, $s(i, j - 1)$, $s(i - 1, j - 1)$, Figure 2 illustrates calculations for every cell based on three previous cells. Furthermore, substitution matrix $S(A_i, B_j)$ must be considered in order to predict biological relationship between two residues.

In the first case the scores $s(i, j)$ is the sum of the score of 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. These processes can be clarified formally by the following equation:

$$\begin{aligned} s(i, j) = \max \{&(s(i-1, j-1) + S(Ai, Bj) \\ &\text{the first case @ } s(i-1, j) - g \\ &\text{the second case @ } s(i, j-1) - g)\} \end{aligned} \quad (1)$$

In alignment of two sequences, mismatches residues can be decreased to zero by inserting more gaps between residues composing of the two sequences. However, insertions of arbitrary gaps affect the biological relevance of sequences. Thus, numerous notable

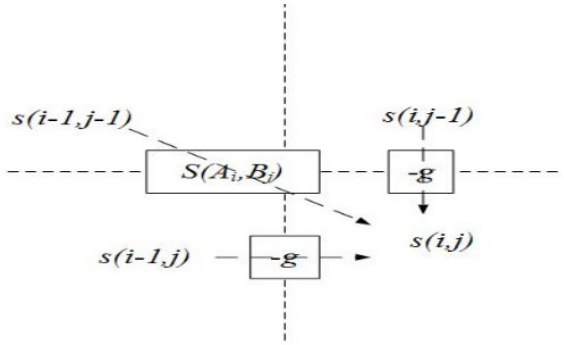

Figure 2. Calculation of s(i, j) based on three previously cells in the matrix.

algorithms have considered penalizing for using gaps in an alignment[3–5,8,9,42,47–52,55]. The most noted models for gaps are a linear gap model and affine gap model. In a linear gap model, each insertion or deletion involves a single residue. A simple way to declare a linear gap penalty function (μ) of a gap length (r) is as follow:

$$\mu(r) = -\delta r \tag{2}$$

where, r is the length of the gaps, and δ is the penalty for each single residue. Penalizing a single residue is impractical in the case of a flood of consecutive gaps of length r. An alternative approach is necessary to penalize consecutive gaps. On another model, the affine gap model consists of two parameters, a gap-opening penalty δ for the first gap residue and a gap extension penalty α for any succeeding gap residues. Equation of a gap penalty using affine gap model is written as follows:

$$\mu(r) = -\delta - \alpha(r-1) \tag{3}$$

Gap extension penalty normally smaller than a gap opening penalty ($\alpha < \delta$), which gives a little cost for long insertions. Affine gap model distinguishes than other gaps models in keeping track of the previous pairing in the alignment, to determine whether a current gap is part of a longer indel or not.

Part 3: Traceback: The best alignment for sequences can be obtained simply by introducing backtracking pointers into the matrix $s(i, j)$. Every back pointer ($*\uparrow\leftarrow$) specify which of the above rules are used to computes $s(i, j)$. Next we reconstruct alignment by traversing the matrix s from the point (n, m) to $(0, 0)$ towards backtracking pointers. When we encounter * then A_i matches to B_j, for $\leftarrow$ we put a gap in the sequence B, and for $\uparrow$ we put a gap in the sequence A.

4. Parallel Implementation of SW Algorithm

The SW algorithm shows serious complications in computing resources, which involve a huge amount of shared processor in parallel architecture. There remains a need for powerful algorithm(s) to utilize this architecture to minimize days and/or hours of calculating similar region in comparing sequences. Much research in recent years has focused on parallel SW algorithms[4,7,13,23,37,51–57]. Unfortunately, these methods have their own drawbacks by ignoring or overlooking load balance techniques, CPU complications in waiting-queue, and communication delay for messages delivery between shared processors in parallel architecture. This paper proposes an algorithm for implementing the SW algorithm in scheduler-workers model called Coarse Grain Load Balance Algorithm (CGLBA).

Data-dependency in sequence alignment involves an algorithm for accelerating the calculation at every worker. Furthermore, reducing waiting-time in data-dependency improves the overall system throughput in sequence alignment problem. In scheduler-workers model, dependent data causes delaying in communication between shared workers. Therefore, the proposed scheduling algorithm considers communication delaying in it implementation. In this condition, the implementation of sequence alignment in a parallel platform is a set of sequential communicating workload. The execution of workloads in a parallel system causes communication delays if the predecessor and the successor workload are executed on different workers.

In this paper, a scheduling algorithm based on Round Robin (RR) algorithm with First Come First Serve (FCFS) technique is developed to accelerate the comparison for long sequences length and to improve the system throughput. At the simplest formulation of the problem, workers are assumed to be connected in a cluster and the predecessor and successor workloads are executed on different workers. Workers compute, send, and receive dependent workloads in RR algorithm, Furthermore, on the completion of a workload the results are broadcast to the successor. However, waiting-time and communication delay occurred when a worker sends a workload to another worker and discontinue until it receives another workload form a predecessor worker[58]. Therefore, FCFS technique is adopted at every worker to reduce the time and communication delay.

In sequence alignment problem, executing workload on a worker makes CPU accumulate a dependent data. Therefore, the time needed to manipulate this data is increasingly affected by the state of the CPU. The ready state is the most desirable case in dependent data in order to reduce waiting time and communication delay. Non-preemptive scheduling algorithm is highly recommended in such cases. Generally, coarse-grain workload is distributed equally to shared workers and involves a small execution time with minimum overhead for transferring workload. In CGLBA, blocks are divided statically and equally between all shared workers based on RR algorithm. Every worker performs SW algorithm and sends

dependent cell to the successor worker, see Figure 3. If the last worker reaches, then the scheduler gathers all results from all workers. Furthermore, scheduling technique with priority based on FCFS algorithm is adopted to reduce waiting time in any worker. The algorithm considers both communications among workers and computation time at every worker. In addition, the coarse grain of blocks involves less communication, which sustains in minimizing communication delay between shared workers. Despite of similarity detection in DNA and/or Protein sequences, this algorithm is valid to address any problems related to data-dependency.

For comparing two sequences a scoring matrix is established and calculated using SW algorithm. Furthermore, blocks of compared sequences with consecutive elements each are broadcasted in a cyclic distribution for all shared workers in scheduler-workers model. The block-wise paradigm distributes the scoring matrix S into $(s_1, s_2, \ldots s_n)$ evenly to blocks of length n with consecutive elements in each block, where n is the number of shared workers. Every block is assigned to symmetric workers, for instance s_n is assigned to worker n. The remaining elements in the last block are assigned to the last worker. The cyclic distributing of the blocks to the shared workers is based on RR algorithm in both processors and successor. Therefore, a row i with elements $(i_1, i_2, \ldots i_j)$ is calculated on the worker i, where $0 < j < n$. On the other hand, waiting time in data-dependency can be addressed by using the priority technique, because for every workload with predecessor and successors workloads stay in a waiting-queue of the CPU. In the scheduling algorithm with priority technique, each workload is assigned to different priority grade in FCFS list. The workload with highest priority is allocated and preempted the CPU. Higher priority processes execute faster than lower priority processes. Waiting time and response time are always smaller, while throughput time depends on the size of the workload. Furthermore, in FCFS list, the CPU utilizes all times with fair execution for all workloads. In addition, no conflict occurs to the workload, because the workload preempt the CPU until it finish. Pseudo code for the first algorithm defines equal partition sizes, while CGLBA consider in the next step as follows

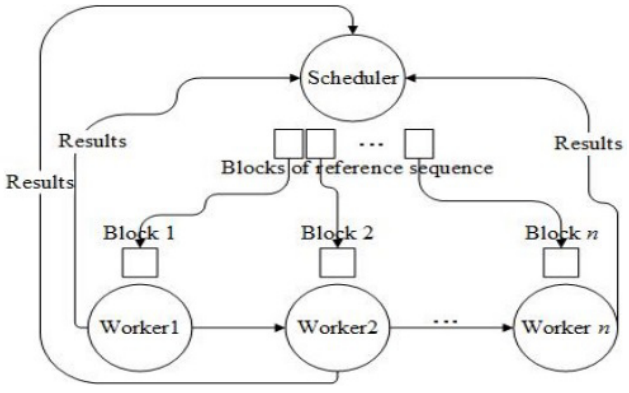

Figure 3. Scheduler sends blocks of reference sequence to all workers evenly.

Algorithm 1: **/*Get each equal partition size to distribute for every worker*/**

```
Get_Worker_Partition(MyId, Total_Workers, Sequence_Size)
1:  Begin
2:      Load_Main = Sequence_Size / Total_Workers;
3:      Load_Mod = (Sequence_Size % Total_Workers) > MyId;
4:      Each_Partition = Load_Main + Load_Mod;
5:      Return Each_Partition;
6:  End
```

Algorithm 2: **/*Distribution of A&B to scheduler and every worker*/**

```
Coarse_Grain_Load_Balance_Algorithm()
Begin
    For(k=0;k<=N;k++) // N is number of workers
    {
    Back_Grnd_Proc = 'Process_Mode_Background_Begin';
    Err_Back_Proc = 'Error_Process_Mode_Already_Background';
    If (!Set_PriorityClass(Get_Current_Process(), Back_Grnd_Proc))
    {
    Get_Info(); /*Get the information of existing process*/
    If (Err_Back_Proc == Error)
        Print ("Failed to enter background mode"), Error);
```

(*Continued*)

Algorithm 2: (*Continued*)

```
        Else
                Print ("Already in background mode");
          }
        Set_Priority_Class (Get_Current_Process(),
        High_Priority_Class);
        /*Display FCFS priority class*/
        CPU_Pri_Class = Get_Priority_Class(Get_
        Current_Process());
        Print ("Current priority class is 0x80Which is a
        High_Priority_Class"), CPU_Pri_Class);
        MyBlock_From_Sequence_A = Get Algorithm 1:
        For (i=1;i<= Sequence_B;i++)
        {
                For (j=1;j<=MyBlock_From_
                Sequence_A;j++)
                {
                Do Smith-Waterman_Algorithm();
                Send last cell from the current block to
                the next worker
                } // End For j
        } // End For i
        } // End For k
End
```

5. Results and Discussion

In most parallel algorithm, execution time is the primary concern. It analyzes the performance and the overall system throughput. The main intention of this section is to evaluate and test CGLBA performance. 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++ for serial execution, furthermore 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 Proteins sequences used in the experiment is obtained from National Center for Biotechnology Information[67] using CLC Sequence Viewer, the GUI bioinformatics software environment. Twenty sequences are considered as datasets listed in Table 1. For comparing results, different sizes of sequences are obtained ranging from 411 KBP to 4 MBP.

Processing times for different compared sequences is calculated, while experiments run on various cores many times for the purpose of accuracy and comparable results. Parameters δ, ε, and λ set to –0.79, –2.8, and –1.34 where max optimal alignment is obtained under these values[65,66].

Table 1. Dataset sizes based on real sequences obtained from NCBI

NCBI Reference	BP Size	ID	NCBI Reference	BP Size	ID
NG_008935.1	411120	N_1	KI635555.1	418695	N_2
NZ_AYWV01000002.1	804182	N_3	NZ_AYXC01000002.1	820688	N_4
NZ_KI542646.1	1209871	N_5	NZ_AYXF01000002.1	1273972	N_6
KI630004.1	1613596	N_7	NZ_AYWE01000001.1	1617746	N_8
GK000039.2	2035851	N_9	GK000056.2	2051248	N_{10}
GK000047.2	2415845	N_{11}	KI632495.1	2476485	N_{12}
NZ_GG704599.1	2868439	N_{13}	GK000038.2	2883116	N_{14}
NW_006203686.1	3223788	N_{15}	NW_006199822.1	3283941	N_{16}
NW_006204120.1	3606379	N_{17}	NC_022997.1	3653837	N_{18}
KI632513.1	4039244	N_{19}	GK000052.2	4072301	N_{20}

CGLBA shows better processing time in the overall system. For all compared sequences, elapsed time seems to linear decrease when more workers are considered, see Figure 4. Serial times are calculated using one worker with different aligned sequences length and it shows highest values as expected. For instance, minimum sequence length 411 × 418 KBP requires 2.56 hours on a serial machine, while the maximum sequence length 4 × 4 MBP requires one day plus, which is seriously unreasonable and a bottleneck. Thus, the need for parallel implementations stems from this point, where for long sequences lengths, days of waiting results using SW remain a natural corollary. However, execution times drop down significantly while using up to eight workers. Furthermore, the best results are obtained by worker eight, where the amount of distributed blocks of reference sequence reaches less values as 52 KBP for sequence with length 4 MBP; minimum time is calculated for the lower sequences with length 411 × 418 KBP and requires 50 minutes, while the maximum time is observed for 8 hours, when calculating sequences with length 4 × 4 MBP.

Generally, execution time increases dramatically in a linear scene when the sequence length also increases. However, comparing with serial machine, execution time decreases significant using CGLBA. Next subsection discusses some performance metrics and comparing results to similar researches.

5.1 Speedup

Speedup measures performance achieved by parallelizing SW algorithm over sequential implementation. Three commonly performance models are always used for measuring speedup metrics include Amdahl[59], Gustafson[60], and Sun and Ni[61]. The speedup S_p of p workers is calculated using the following equation:

$$s_p = \frac{T_s}{T_p} \tag{4}$$

where, T_s and T_p are serial and parallel time respectively.

The speedup values are calculated by equation 8. As shown in Figure 5 valuable results are gained using CGLBA. For instance, comparing longest datasets N19 × N20 on eight workers achieves speedup of 1.18, 1.40, 1.64, 1.97, 2.47, 2.85, and 3.31.

The experimental results show that CGLBA gains fair speedup while using four workers, however best results of speed up is gained while using five workers and above. In general, proportional relationship appears between increasing in speed up and the number of workers. However, for less sequences length speed up shows low values, while for high sequence length speed up growths with highest values. Due to evenly blocks distributed for all shared workers, close values of speedup is recorded for different set of sequences run on a cluster of eight workers. These close values also promote consistency of experimental architecture while running the algorithm.

5.2 Efficiency

Efficiency estimates utilization of workers in solving CGLBA compared to communication and synchronization complexity. It is measured in term of resource consumption. The measure taken as a ratio between speed up S_p and the number of workers p as follows:

$$E_p = \frac{S_p}{p} = \frac{T_s}{pTp} \tag{5}$$

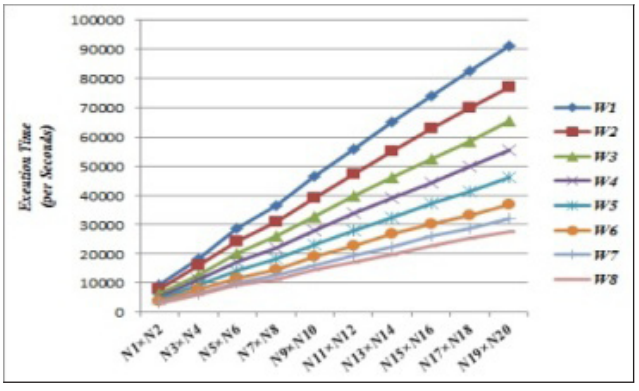

Figure 4. Execution time calculated using eight workers with a different dataset sizes.

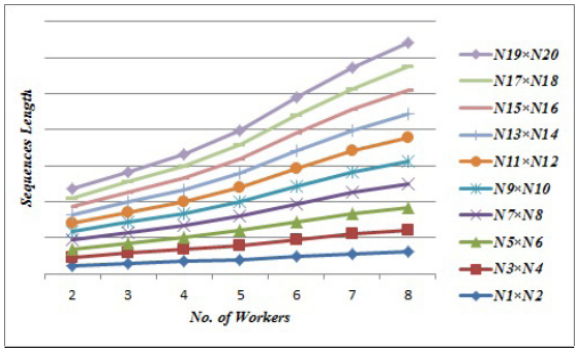

Figure 5. Close values of speedup for 10 sets of sequences with different lengths run on a cluster of eight workers.

where, T_s denotes serial time and T_p indicates processing time for p workers.

Efficiency values typically between zero and one, while speedup between zero and the number of workers[61,62]. The efficiency of the proposed algorithm is calculated on eight workers E2-E8 with different sizes of datasets, see Figure 6. In most cases, efficiency achieves the values of 0.57, 0.58, and 0.59 when using two workers. However, 0.38, 0.39, 0.40, and 0.41 are achieved using eight workers. Furthermore, the efficiency increases in the case of incremental of dataset sizes. As can be seen from Figure 6, the efficiency of CGLBA is slightly decreases while the number of workers increases, which could be due to increasing communication and synchronization overhead.

5.3 Scalability

Scalability indicates an efficiency of CGLBA by measuring the capacity to effectively harness as the number of workers increases. It measures increasing of speedup in proportion to the number of workers. Scalability is used to predict performance of CGLBA as well as the parallel architecture for large datasets; it determines variability of speedup S and efficiency E while increasing workers P for different datasets sizes[63,64]. Measurements of scalability using speedup and efficiency can be observed in Figure 7.

To objectively evaluate the performance of CGLBA, it is tested against similar works executed in the same platform in the context of multiple workers. Batista et al.[4] proposed z-align, a parallel strategy to reduce time and space needed to parallel local alignments for large sequences. The algorithm is implemented in four phases: (1) distribute compared sequences to all processors, (2) calculated the similarity matrix, (3) collect results with best score, and (4) produce optimal local alignment(s). While, Nordin et al.[37] develope FRA-Search model to improve comparisons of large DNA sequence; two approaches are used in the model: string matching algorithm and rough sets theory. In order to compare speedup results the test-bed considered z-align, FRA-Search, and CGLBA using 400K × 400K nucleotides. As illustrated in Figure 8, CGLBA achieves significant speedup, while the number of nucleotides in the comparison is bigger than that consider in other researches.

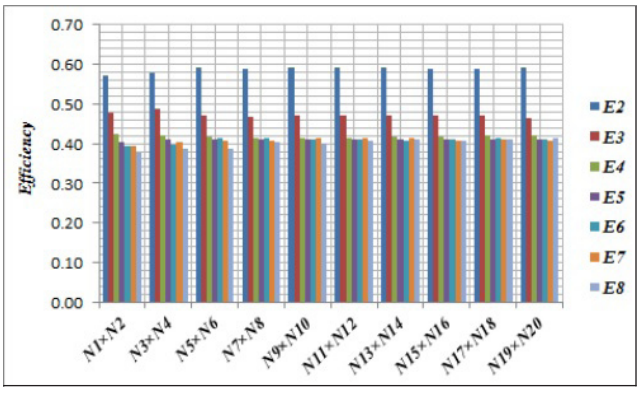

Figure 6. Efficiency of CGLBA using eight workers E2-E8 and different sizes of datasets.

6. Conclusion

Studying SW algorithm is a challenging area for a number of years. Since 1981 and till now, SW algorithm providing optimal solution with accurate results when comparing

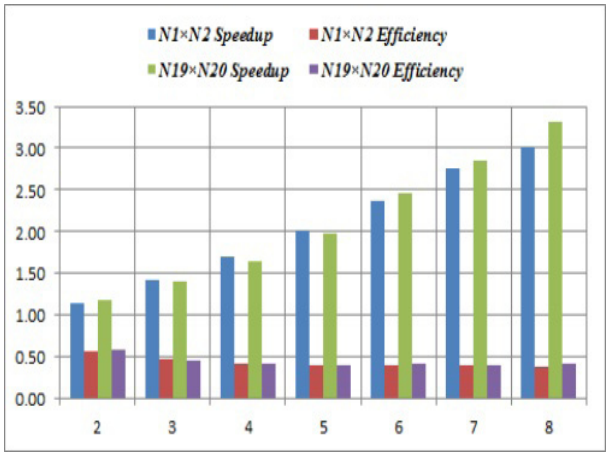

Figure 7. Increase of speedup and slight decreasing of efficiency using two different datasets sizes N1 × N2 and N19 × N20 run on eight workers.

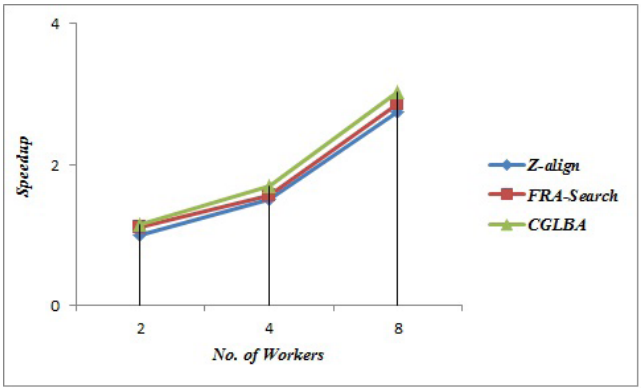

Figure 8. Speedup of z-align, FRA-Search, and CGLBA.

sequences. However, the major drawback is the complexity in comparing long sequences. In attempt to reduce consumption time in SW, an algorithm is designed, developed, and implemented in this paper called CGLBA. In CGLBA, blocks are divided statically and equally between all shared workers based on RR algorithm with FCFS technique, where each worker performs SW; finally the scheduler gathers all results from all workers. Performance and evaluation of CGLBA as well as a measurement of performance metrics are discussed in this paper, which included speedup, efficiency, scalability, and implementation cost. For comparison of longest datasets N19 × N20, achieves speedup 1.18, 1.40, 1.64, 1.97, 2.47, 2.85, and 3.31 in eight workers. Moreover, in most cases, efficiency achieves 0.57, 0.58, and 0.59 using two workers. Scalability is measured in term of speedup and efficiency using two different datasets sizes N1 × N2 and N19 × N20 run on eight workers. The results show fair scalability is obtained by increasing of speedup and slight decreasing of efficiency. Further researches are needed to better understand the impact of memory mapping in scheduler-workers model, and it is effect on the communication costs and/or space complexity. This future research should, therefore, concentrates on applicable parallel model for DM and SDM with multiprocessor's architecture.

7. References

1. Axelson-Fisk M. Sequence alignment. Comparative Gene Finding. 2010; 89–155
2. Brenner S. Optimal pairwise alignment; 2011.
3. Díaz D, Esteban FJ, Hernández P, Caballero JA, Dorado G, Gálvez S. Parallelizing and optimizing a bioinformatics pairwise sequence alignment algorithm for many-core architecture. Parallel Comput. 2011
4. Batista RB, Boukerche A, de Melo ACMA. A parallel strategy for biological sequence alignment in restricted memory space. J Parallel Distr Comput. 2008; 68(4):548–61.
5. Boukerche A, de Melo ACMA, Sandes EFO, Ayala-Rincon M. An exact parallel algorithm to compare very long biological sequences in clusters of workstations. Cluster Computing. 2007; 10(2):187–202.
6. Guo T, Li G, Deaton R. Accelerating Computation of DNA Sequence Alignment in Distributed Environment. Advanced Intelligent Computing Theories and Applications. With Aspects of Contemporary Intelligent Computing Techniques. 2008; 222–28.
7. Nordin M, Rahman A. Utilizing MPJ Express Software in Parallel DNA Sequence Alignment; 2009.
8. Needleman SB, Wunsch CD. A general method applicable to the search for similarities in the amino acid sequence of two proteins. J Mol Biol. 1970; 48(3):443–53.
9. Smith T, Waterman M. Identification of common molecular subsequences. J. Mol. Bwl. 1981; 147:195–97.
10. Pearson WR, Lipman DJ. Improved tools for biological sequence comparison. Proceedings of the National Academy of Sciences. 1988; 85(8):2444.
11. Altschul SF, Gish W, Miller W, Myers EW, Lipman DJ. Basic local alignment search tool. J Mol Biol. 1990; 215(3):403–10.
12. Hasan L, Al-Ars Z, Vassiliadis S. Hardware acceleration of sequence alignment algorithms-an overview; 2007.
13. Borovska P, Lazarova M. Parallel models for sequence alignment on CPU and GPU; 2011.
14. Manavski S, Valle G. CUDA compatible GPU cards as efficient hardware accelerators for Smith-Waterman sequence alignment. BMC bioinformatics. 2008; 9(Suppl 2):S10.
15. Schatz M, Trapnell C, Delcher A, Varshney A. High-throughput sequence alignment using Graphics Processing Units. BMC bioinformatics. 2007; 8(1):474.
16. Zhang Y, Misra S, Honbo D, Agrawal A, Liao W, Choudhary A. Efficient pairwise statistical significance estimation for local sequence alignment using GPU; 2011.
17. Allred J, Coyne J, Lynch W, Natoli V, Grecco J, Morrissette J. Smith-Waterman implementation on a FSB-FPGA module using the Intel Accelerator Abstraction Layer. IEEE International Symposium on Parallel & Distributed Processing, IPDPS; 2009.
18. Meng X, Chaudhary V. Boosting data throughput for sequence database similarity searches on FPGAs using an adaptive buffering scheme. Parallel Computing. 2009; 35(1):1–11.
19. Yilmaz C, Gök M. System designs to perform bioinformatics sequence alignment. Turk J Electr Eng Comput Sci. 2013; 21:246–62.
20. Sarkar S, Kulkarni GR, Pande PP, Kalyanaraman A. Network-on-chip hardware accelerators for biological sequence alignment. IEEE Trans Comput. 2010; 59(1):29–41.
21. Jin HQ, Jespersen D, Mehrotra P, Biswas R, Huang L, Chapman B. High performance computing using MPI and OpenMP on multi-core parallel systems. Parallel Computing. 2011; 37(9):562–75.
22. Trelles O. On the parallelisation of bioinformatics applications. Briefings in Bioinformatics. 2001; 2(2):181.
23. Bandyopadhyay S, Mitra R. A parallel pairwise local sequence alignment algorithm. IEEE Transactions on NanoBioscience. 2009; 8(2):139–46.
24. Delgado G, Aporntewan C. Data dependency reduction in Dynamic Programming matrix. Eighth International Joint Conference on Computer Science and Software Engineering (JCSSE); 2011.

25. Sebastião N, Encarnação G, Roma N. Implementation and performance analysis of efficient index structures for DNA search algorithms in parallel platforms. Concurrency and Computation: Practice and Experience; 2012.
26. Borovska P, Gancheva V, Dimitrov G, Chintov K. Parallel performance evaluation of multithreaded local sequence alignment; 2011.
27. Ivan G, Banky D, Grolmusz V. Fast and exact sequence alignment with the Smith-Waterman algorithm: the swissalign webserver. arXiv preprint arXiv:1309.1895; 2013.
28. Mendonca FM, Melo ACMA. Biological sequence comparison on hybrid platforms with dynamic workload adjustment. Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2013 IEEE 27th International; 2013.
29. Neves N, Sebastiao N, Patricio A, Matos D, Tomás P, Flores P, Roma N. BioBlaze: Multi-core SIMD ASIP for DNA sequence alignment. 2013 IEEE 24th International Conference on Application-specific Systems, Architectures and Processors (ASAP); 2013.
30. Satyanvesh D, Balleda K, Baruah P. Genalign—A high performance implementation for aligning the compressed DNA sequences. 2013 15th International Conference on Advanced Computing Technologies (ICACT); 2013.
31. Satyanvesh D, Balleda K, Padyana A, Baruah P. GenCodex-a novel algorithm for Compressing DNA sequences on multicores and GPUs. 19th IEEE International conference on High Performance Computing; 2012 Dec.
32. Alachiotis N, Berger S, Flouri T, Pissis SP, Stamatakis A. Libgapmis: extending short-read alignments. BMC Bioinformatics. 2013; 14(Suppl 11):S4.
33. Pérez HM, Tárraga J, Medina I, Barrachina S, Catalán MIC, Dopazo J, Orti ESO. Concurrent and Accurate RNA Sequencing on Multicore Platforms; 2013.
34. Chorley MJ, Walker DW. Performance analysis of a hybrid MPI/OpenMP application on multi-core clusters. Journal of Computational Science. 2010; 1(3):168–74.
35. Noorian M, Pooshfam H, Noorian Z, Abdullah R. Performance enhancement of smith-waterman algorithm using hybrid model: comparing the MPI and hybrid programming paradigm on SMP clusters. IEEE International Conference on Systems, Man and Cybernetics, SMC; 2009.
36. Luecke KR. Software development for parallel and multicore processing: InTech; 2012.
37. Nordin A, Yazid M, Aziz A, Osman M. Parallel guided dynamic programming approach for DNA sequence similarity search; 2009.
38. Dudley JT, Butte AJ. A quick guide for developing effective bioinformatics programming skills. PLoS Comput Biol. 2009; 5(12):e1000589.
39. Zhang X, Zhou X, Wang X. Basics for bioinformatics basics of bioinformatics: Springer; 2013.
40. Hirschberg DS. A linear space algorithm for computing maximal common subsequences. Communications of the ACM. 1975; 18(6):341–43.
41. Huang X, Hardison RC, Miller W. A space-efficient algorithm for local similarities. Computer applications in the biosciences: CABIOS. 1990; 6(4):373–81.
42. Myers EW, Miller W. Optimal alignments in linear space. Computer applications in the biosciences: CABIOS. 1988; 4(1):11–17.
43. Wu C, Kalyanaraman A, Cannon WR. A scalable parallel algorithm for large-scale protein sequence homology detection. 39th International Conference on Parallel Processing (ICPP). 2010.
44. Steinfadt S. Fine-grained parallel implementations for SWAMP+ Smith–Waterman alignment. Parallel Computing. 2013; 39(12):819–33.
45. Castaño FS, Ramirez A, Valero M. Quantitative analysis of sequence alignment applications on multiprocessor architectures; 2009.
46. Sánchez F, Cabarcas F, Ramirez A, Valero M. Long DNA sequence comparison on multicore architectures. Euro-Par 2010-Parallel Processing. 2010; 247–59.
47. Chen C, Schmidt B. An adaptive grid implementation of DNA sequence alignment. Future Generat Comput Syst. 2005; 21(7):988–1003.
48. Gotoh O. An improved algorithm for matching biological sequences. (J Mol Biol. 1982; 162(3):705–08.
49. Hosangadi S, Kak S. An alignment algorithm for sequences. arXiv preprint arXiv:1210.8398; 2012.
50. Hudek A, Brown D. FEAST: sensitive local alignment with multiple rates of evolution. IEEE ACM Trans Comput Biol Bioinformatics. 2011; 99:1.
51. Rajko S, Aluru S. Space and time optimal parallel sequence alignments. IEEE Transactions on Parallel and Distributed Systems. 2004; 15(12):1070–81.
52. Zhang F, Qiao X-Z, Liu Z-Y. A parallel smith-waterman algorithm based on divide and conquer. Fifth International Conference on Algorithms and Architectures for Parallel Processing; 2002
53. Boukerche A, de Melo A, Ayala-Rincon M, Santana T. Parallel smith-waterman algorithm for local DNA comparison in a cluster of workstations. Experimental and Efficient Algorithms. 2005; 131–44.
54. Boukerche A, de Melo ACMA, Walter MET, Melo RCF, Santana MNP, Batista RB. A performance evaluation of a local dna sequence alignment algorithm on a cluster of workstations; 2004.
55. Boukerche A, de Melo ACMA, Ayala-Rincón M, Walter MEMT. Parallel strategies for the local biological sequence alignment in a cluster of workstations. Journal of Parallel and Distributed Computing. 2007; 67(2):170–85.

56. Du Z, Ji Z, Lin F. Parallel computing for optimal genomic sequence alignment. Fuzzy Systems and Knowledge Discovery. 2006; 532–35.
57. Meng X, Chaudhary V. Exploiting multi-level parallelism for homology search using general purpose processors; 2005.
58. Drozdowski M. Scheduling with communication delays Scheduling for Parallel Processing. Springer; 2009.
59. Amdahl GM. Validity of the single processor approach to achieving large scale computing capabilities. Proceedings of the spring joint computer conference; 1967 Apr 18–20.
60. Gustafson JL. Reevaluating Amdahl's law. Comm of the ACM. 1988; 31(5):532–33.
61. Sun X-H, Ni LM. Another view on parallel speedup. Proceedings of Supercomputing'90; 1990.
62. Shi Y. Reevaluating Amdahl's law and Gustafson's law. Computer Sciences Department, Temple University (MS: 38–24); 1996.
63. Bell G. Scalable, parallel computers: alternatives, issues, and challenges. Int J Parallel Program. 1994; 22(1):3–46.
64. Kumar V, Gupta A. Analysis of scalability of parallel algorithms and architectures: a survey. Proceedings of the 5th international conference on Supercomputing; 1991.
65. Durbin R. Biological sequence analysis: probabilistic models of proteins and nucleic acids: Cambridge University Press; 1998.
66. Holmes I. Studies in probabilistic sequence alignment and evolution. Queens' College; 1998
67. NCBI. National Center for Biotechnology Information. 2014 15 Feb. Available from http://www.ncbi.nlm.nih.gov/
66. Rognes T. Faster Smith-Waterman database searches with inter-sequence SIMD parallelisation. BMC bioinformatics. 2011; 12(1):221.