Available online at www.sciencedirect.com
ScienceDirect
J. Parallel Distrib. Comput. 67 (2007) 1007 – 1017
Journal of Parallel and Distributed Computing
www.elsevier.com/locate/jpdc

On the design of high-performance algorithms for aligning multiple protein sequences on mesh-based multiprocessor architectures

Diana H.P. Low[a], Bharadwaj Veeravalli[a,*], David A. Bader[b]

[a]Department of Electrical and Computer Engineering, The National University of Singapore, 4 Engineering Drive 3, Singapore 117576, Singapore
[b]College of Computing, Georgia Institute of Technology, Atlanta, Georgia, USA

Received 5 June 2006; received in revised form 8 February 2007; accepted 22 March 2007
Available online 17 May 2007

Abstract

In this paper, we address the problem of multiple sequence alignment (MSA) for handling very large number of proteins sequences on mesh-based multiprocessor architectures. As the problem has been conclusively shown to be computationally complex, we employ divisible load paradigm (also, referred to as divisible load theory, DLT) to handle such large number of sequences. We design an efficient computational engine that is capable of conducting MSAs by exploiting the underlying parallelism embedded in the computational steps of multiple sequence algorithms. Specifically, we consider the standard Smith–Waterman (SW) algorithm in our implementation, however, our approach is by no means restrictive to SW class of algorithms alone. The treatment used in this paper is generic to a class of similar dynamic programming problems. Our approach is recursive in the sense that the quality of solutions can be refined continuously till an acceptable level of quality is achieved. After first phase of computation, we design a heuristic scheme that renders the final solution for MSA. We conduct rigorous simulation experiments using several hundreds of homologous protein sequences derived from the *Rattus Norvegicus* and *Mus Musculus* databases of olfactory receptors. We quantify the performance based on speed-up metric. We compare our algorithms to serial or single machine processing approaches. We testify our findings by comparing with conventional equal load partitioning (ELP) strategy that is commonly used in the parallel processing literature. Based on our extensive simulation study, we observe that DLT paradigm offers an excellent speed-up characteristics and provides avenues for its use in several other biological sequence processing related problem. This study is a first time attempt in using the DLT paradigm to devise efficient strategies to handle large scale multiple protein sequence alignment problem on mesh-based multiprocessor systems.
© 2007 Elsevier Inc. All rights reserved.

Keywords: Mesh topology; Smith–Waterman algorithm; Multiple sequence alignment; Divisible loads; Protein sequences

1. Introduction

Biological sequence alignment has significance in understanding genetic and molecular evolution. Biological sequences are made up of residues. In DNA (deoxyribonucleic acid) sequences, these residues are nucleic acids, while in protein sequences, these residues are amino acids. Aligning or comparing biological sequences is a computationally intensive operation as we need to take into consideration that biological sequences may mutate or evolve over time. The process of aligning two or more sequences is often an imperative step to quantify the quality of the samples under consideration. For instance in case of protein structure predictive methods and structure comparison methods, sequence alignment for maximum similarity score is often one of the crucial steps [10]. As such, in aligning biological sequences, residues can be inserted, deleted or substituted from either two sequences to obtain the optimum alignment. For two sequences with x number of residues each, there are as much as $(1+\sqrt{2})^{2x+1}\sqrt{x}$ possible alignment combinations [32]. Hence, over the years various algorithms have been proposed in order to speed up the alignment process.

Protein sequence analysis, unlike DNA sequence analysis, tries to match protein sequences and find conserved domains within the sequences in order to classify them into families of similar functions. Multiple sequence alignment (MSA) analysis provides a wealth of information; protein structure, active domain sites and protein binding sites can be revealed through such analysis. In 1970, Needleman and Wunsch [20] have

* Corresponding author. Fax: +65 777 8804.
E-mail addresses: dianalow@nus.edu.sg (D.H.P. Low), elebv@nus.edu.sg (B. Veeravalli), bader@cc.gatech.edu (D.A. Bader).

0743-7315/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.jpdc.2007.03.007

designed an algorithm for aligning biological sequences without going through all the possible combinations. Nevertheless, the Needleman–Wunsch algorithm still has a complexity of $O(x^2)$. This algorithm is later improved by Sellers [26] and generalized by Smith and Waterman [27,33]. The Smith–Waterman (SW) algorithm, on the other hand, has a complexity of $O(x^3)$ but was later improved by Gotoh [13] to just $O(x^2)$. Considering the vast amount of sequences available today from databases such as [9,12,29], a complexity of $O(x^2)$ may still be unacceptable in many cases especially in the case of aligning multiple sequences. Further, these gigantic databases are growing rapidly, i.e., the GenBank is growing at an exponential rate, with rate as much as of 1.2 million new sequences a year [3].

In order to cope with the computationally intensive operation and vast amount of available data, various heuristic methods have been proposed. These include FASTP [16], FASTA [21,22], BLAST [1], variants of BLAST such as mpiBLAST [11] and FLASH [7]. These heuristics obtain computation speed-up at the cost of less sensitivity. Other methods, such as [19], are able to achieve speed-ups without losing sensitivity. Nevertheless, the algorithms are only effective in aligning similar sequences as the computational time is directly proportional to the number of differences in between the sequences. Further, they do not generate the complete SW matrix that can be used to detect multiple subsequence similarities.

With the recently evolved cluster/grid computing paradigm, researchers attempt to gain high speed-ups in handling such computationally intensive applications. In [36], a speculative strategy was presented for multi-sequence alignment. The strategy exploits the independence between alignment group pairs in the Berger–Munson [4] algorithm. It achieves speed-up by processing multiple iterations in parallel with the *speculation* that the current iteration is independent of the previous iteration. Nevertheless, due to the working style of this strategy, the number of processors that can be utilized is limited and the speed-up is dependant on the similarities of the sequences.

In [31], a clustering strategy was introduced for multi-sequence alignment. In this strategy, sequences are cleverly filtered and group pairs identified. These group pairs are then processed concurrently hence achieving speed-up. Nonetheless, in this strategy, there are relatively large amount of idle processors during the early stages of the strategy when the number of group pairs that can be processed in parallel are limited. Further, homogenous processors are required in for this strategy for high degree of parallelism.

In a recent work [25], a parallel strategy was introduced that utilizes the advantages offered by the generic Intel processor with MultiMedia eXtensions (MMX) and Streaming SIMD Extensions (SSE) technology. As shown in the paper, the strategy utilizes the technology offered by the processor to execute eight computational processes in parallel. The major disadvantage of this strategy is that it is nonscalable and dependent on the micro-processor technology, i.e., the Pentium processor shown in the paper can only support a maximal of eight simultaneous computations. The human genome project has made tremendous progress in sequencing the genomes, and large amount of new sequences are being generated everyday. There is a need to improve processing time and also the quality of the results. Parallel and/or distributed computing can certainly help to reduce the load by disseminating the total computational process among several nodes on the network [37]. This has been attempted by several authors in this bioinformatics domain [23,25]. Divisible load theory (DLT) [6] offers quality solution at a minimum time by partitioning the total load to be processed among several nodes. DLT is proven to be suitable for handling large scale computational loads. In DLT computational loads are divided among the nodes [24], however, the question of which of the entities must be considered as divisible loads is the first step to be decided. In our context, the computational space (which will be described in Section 2) is divided among the nodes as opposed to dividing a protein sequence among the nodes. This is mainly because of the dependencies arising among the processed results on different nodes, if the sequences were partitioned. Thus, we exploit both the space as well as time dimensions to maximize the throughput (number of sequences that can be processed) in our strategy. In a recent work [34], a bus network topology is considered for aligning two biological sequences and the applicability and the power of DLT model is conclusively demonstrated. A linear speed-up was achieved and a very high utilization of processors in the system were demonstrated in this work. This naturally sets a motivation to use the DLT paradigm to handle multiple sequences and develop an alignment engine for carrying out MSA. The relevance is obvious when network-based computing is to be rendered using modern day Bio-Grid architectures and Clusters. Sections 2 and 3 describe our approach in detail.

1.1. Motivation and our contributions

In this paper, we consider the problem of aligning multiple protein sequences on a tightly coupled mesh-based multiprocessor networks. Our venture in this paper is motivated by the fact that there exists a very few online MSA engines that can handle several large number of sequences. Moreover these engines may not serve as a middle-ware plug-and-play module for infrastructures such as, Bio-Grid and network-based service rendering. Further such engines schedule a given set of sequences only once and do not attempt to refine the quality of solution. This poses a serious limitation for biologists and computational researchers to perform a comparative study involving more than hundreds of sequences. Applications such as drug targeting/manufacturing requires a higher level precision in quality of results. All these factors collectively set a demand in designing an (network-based) alignment engine that can handle very large number of sequences. Mesh architectures being commonly used infrastructure in parallel processing community, ranging for a wide variety of applications, we aim to design a generic massively parallel alignment engine that is capable of handling several sequences using a mesh-based multiprocessor topology. We develop algorithms to achieve significant speed-up in processing time as compared to serial or single machine processing approaches. We testify all our findings by comparing with conventional equal load partitioning

(ELP) strategy that is commonly used in the parallel processing literature. We use DLT paradigm and exploit parallelism in space as well as in time dimensions. Our strategy fundamentally exploits the underlying parallelism embedded in the computational steps of multiple sequence algorithms. We consider the standard SW algorithm in our implementation, however, it may be noted that our approach is by no means restrictive to SW algorithm alone. Our approach can be readily applied to a generic class of such similar dynamic programming problems. After initial phase of computation, we design a heuristic scheme, a modified version of Taylor's methodology [10], that renders the final solution for MSA. Our scheme can be recursively used to continuously refine the scores till a satisfactory result is achieved. We conduct rigorous simulation experiments using several hundreds of homologous protein sequences derived from the *Rattus Norvegicus* and *Mus Musculus* databases of olfactory receptors. This is also a first time in the domain of DLT that such an application involving multiple sequences is explicitly attempted. Finally, it may be noted that although we consider a mesh topology, an underlying physical infrastructure could be a cluster or any HPC system with a group of computer nodes and a logical organization could be a mesh.

2. Preliminary information and problem formulation

We shall now present some of the required technical background material first before we present our approach. Firstly, a variant of SW algorithm [27,35] proposed by Gotoh [13] as well as some characteristics of the matrix generated by the algorithm is discussed. In aligning two sequences *Seq*1 and *Seq*2, of length α and β, respectively, the algorithm fundamentally generates three matrices, **S**, **P** and **Q**. Each row and column of these matrices represents a residue of *Seq*1 and *Seq*2, respectively. Given $s(a_x, b_y)$, the substitution score for replacing the *x*th residue from *Seq*1 with the *y*th residue from *Seq*2, w_1 as the penalty for introducing a gap, and v as the penalty for extending a gap, the **S**, **P** and **Q** matrices are related by the following recursive equations:

$$S_{0,y} = S_{x,0} = P_{0,y} = Q_{x,0} = 0, \tag{1}$$

$$S_{x,y} = \max\left\{P_{x,y},\ \ S_{x-1,y-1} + s(a_x, b_y),\ \ Q_{x,y}\right\}, \tag{2}$$

$$P_{x,y} = \max\left\{S_{x-1,y} + w_1,\ \ P_{x-1,y} + v\right\}, \tag{3}$$

$$Q_{x,y} = \max\left\{S_{x,y-1} + w_1,\ \ Q_{x,y-1} + v\right\} \tag{4}$$

for the range $1 \leqslant x \leqslant \alpha$, $1 \leqslant y \leqslant \beta$ where $S_{x,y}$, $P_{x,y}$, and $Q_{x,y}$ represent the matrix elements in the *x*th row, *y*th column of the matrices **S**, **P** and **Q**, respectively. Residues in *Seq*1 and *Seq*2 are tested for a best possible alignment in a recursive fashion, and leads to a possible alignment of the respective sequences. The score of the matrix element, $S_{x,y}$, quantifies the quality of alignment until the *x*th residue of *Seq*1 and the *y*th residue of *Seq*2. The higher the score at $S_{x,y}$, the better the alignment between the sequences up to those residues. The finer details of this algorithm and an illustrative example demonstrating the generation of the above matrices S, P, and Q can be found in [35].

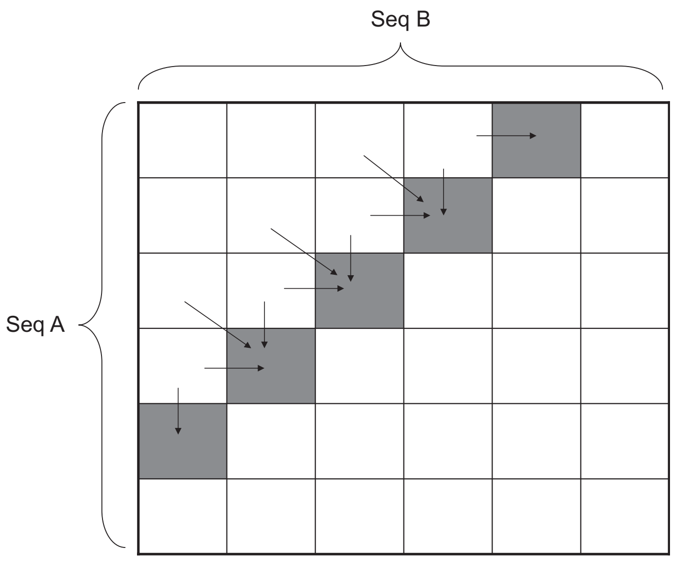

Fig. 1. Computational dependency in generating **S** matrix in Smith–Waterman algorithm.

As can be seen from the equations, the element $S_{x,y}$ is dependent on the $(x-1, y-1)$, $(x-1, y)$, and $(x, y-1)$ elements from the **S**, **P** and **Q** matrices, respectively, shown in Fig. 1. Due to this dependency, the **S** matrix elements cannot be computed independently (either column-wise or row-wise) but the elements along the diagonal line with the same $(x+y)$ values can be computed independently. This property will be exploited in our strategy in the attempt to distribute computations among several processors in the mesh. Secondly, we use Taylor's [28] clustering strategy [1] for multiple protein sequence alignment. It must be stressed that the choice of this strategy is due to its simplicity in implementation and in our strategy, any clustering strategy could be used in general. Basically this strategy makes use of the similarity scores obtained via SW and orders the sequences into a cluster by decreasing similarity scores. Thirdly, divisible load paradigm is employed [5,24] to partition the computational space among the processors in the mesh to enable simultaneous processing in order to achieve higher speed-up. DLT is a methodology that partitions the load arbitrarily to various processors in order to minimize overall processing time. Details on how we use DLT paradigm will be presented in the next section after introducing a description on the underlying mesh network.

We now describe the underlying mesh architecture, as in Fig. 2. We envisage our mesh architecture as a tightly coupled (no communication delays) structure comprising $N \times M$ processing nodes (or processors/CPUs) as shown in the figure. We designate each row as $R_i,\ i = 1, 2, \ldots, N$ and the processors on each row as $P_k, k = 1, \ldots, M$. Each row has a master node that co-ordinates the activities of the processors in that row.

[1] Other alternative methods like, Martinez's [17], Barton and Sternberg's [2] strategies can also be used.

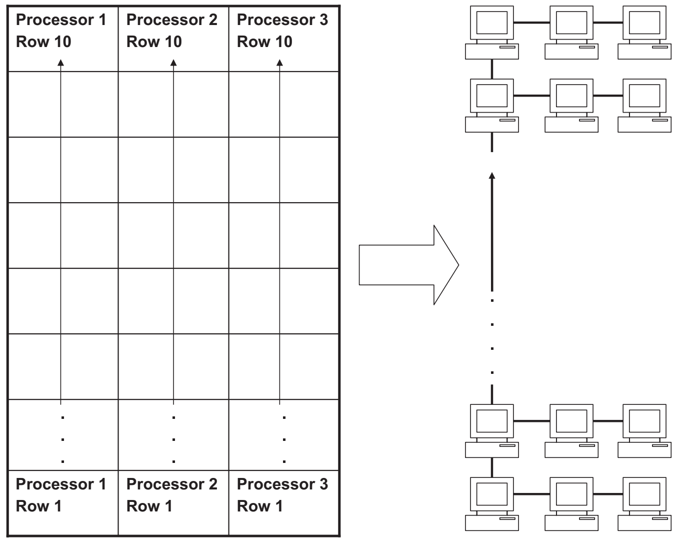

Fig. 2. $N \times M$ mesh structure.

We assume that a process which starts an instance of MSA retrieves a set of sequences to be aligned from a database and injects to the mesh following the distribution strategy to be described in the following subsections. The main sequence pool **O** (see Fig. 3) is divided equally among all the rows so that each row handles the same number of sequences, say, Q. Thus, each row of the mesh will be processing a subset of sequences and arrive at an alignment that is best for that subset. All master nodes of a each row will collate the results to recommend an ultimate aligned sequence set.

In this paper, Taylor's method is considered as a heuristic to align the sequences. Then this set of locally aligned sequences, are passed to another module, referred to as *improved Taylor's method* (ITM), to realign all the locally aligned sequences from each row to improve the overall score. This overall aligned set (only the indices or name tags[2]) is temporarily stored for comparing the quality of the output from the next iteration. We re-designate the sequences following this aligned order and randomly form a set of sequences Q for each row of the mesh for the next iteration for obtaining a refined score. The process is repeated until a satisfactory score and performance gain is achieved. Fig. 3 shows this entire process. It is assumed that all the processors that are involved in the computation of the SW matrices will be furnished with an information on the respective subset of sequences involved, in their local memories. In Table 1 we list an index of notations and terminology that are used throughout this paper.

3. Design and analysis of MSA strategies

3.1. Load distribution strategy

Processing time computation involves capturing the time taken to generate the S, P and Q matrices. All M processors in a row will be involved in the alignment of any one pair of sequences at any particular time. Thus, the matrices will be partitioned into M sections, both in row- and column-wise directions. All three matrices are partitioned into sub-matrices $L_{k,l}, k = 1, \ldots, M; l = 1, \ldots, M$, with each sub-matrix containing a portion of $Seq1$ and $Seq2$ as shown in Fig. 4. Computation starts at L_{11}. Since L_{12} and L_{21} has dependency on L_{11} as described in previous sections (see Fig. 1) they cannot begin computation until L_{11} is completed. This applies to the rest of the matrix entries as well. The effect of this dependency will be seen in the processing time calculation procedure later in this paper.

Following the DLT paradigm, the amount of computational load allocated to each processor will be proportional to its speed, so that all the processes will complete its computation at the same instant. In this paper, the computation space is partitioned as follows. All processors are assigned an identical amount of computational space along $Seq2$ (β/M) following an ELP strategy, whereas the number of rows is derived using DLT paradigm, along $Seq1$ direction. Thus, our aim is to have, on each row R_i,

$$\alpha_k \omega_k = \alpha_{k+1}\omega_{k+1}, \quad k = 1, \ldots, M-1, \tag{5}$$

where

$$\sum_{k=1}^{M} \alpha_k = \alpha. \tag{6}$$

Rewriting (5) in a recursive fashion in terms of α_M, we have,

$$\alpha_k = \alpha_M \left(\frac{\omega_M}{\omega_k}\right). \tag{7}$$

Substituting (7) into (6) we obtain,

$$\alpha_M \left[1 + \sum_{j=1}^{M-1} \left(\frac{\omega_M}{\omega_j}\right)\right] = \alpha,$$

$$\alpha_M = \frac{\alpha}{1 + \sum_{j=1}^{M-1}\left(\frac{\omega_M}{\omega_j}\right)}.$$

Using Eqs. (7) and (8) the net computation space assigned to processor k is

$$\alpha_k = \frac{\alpha}{1 + \sum_{j=1}^{M-1}\left(\frac{\omega_M}{\omega_j}\right)} \left(\frac{\omega_M}{\omega_k}\right). \tag{8}$$

Thus, this effectively completes the task of load distribution to the various processors. ELP is a subset of the DLT problem, in which all the loads are partitioned equally. In this paper, this applies to the division of computation space of $Seq2$, where $\beta_k = \beta_{k+1}$. This means only α_k values will differ while β_k values are constant (for a particular $N \times M$ mesh).

[2] We can store the FASTA name tags associated with each sequence in the order.

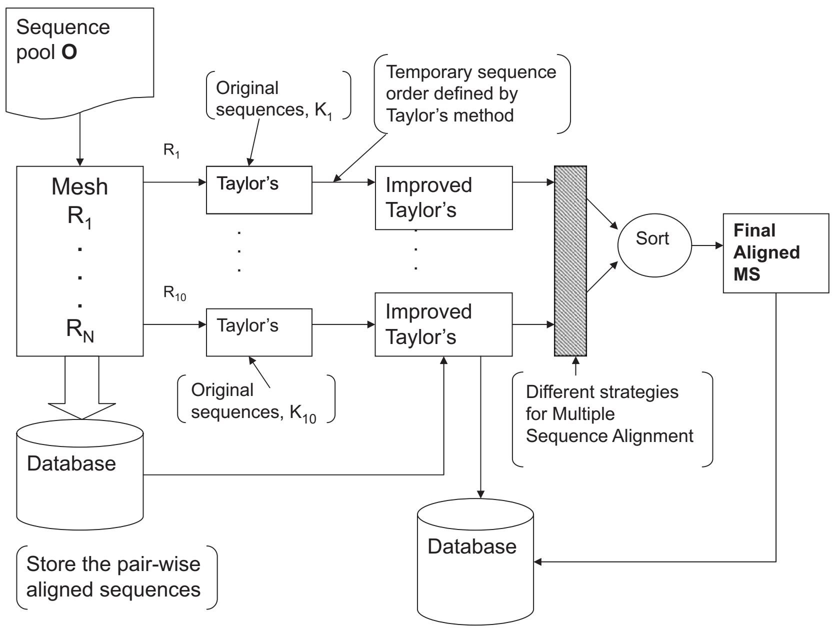

Fig. 3. Design and implementation of the system.

Table 1
Table of definitions

R_i	The row number, $i = 1, \ldots, N$
P_k	kth processor on a row, $k = 1, \ldots, M$
α	Length of $Seq1$ or number of amino acid residues in $Seq1$
β	Length of $Seq2$ or number of amino acid residues in $Seq2$
α_k	Length/computation space of $Seq1$ processed by P_k, where $\sum_{k=1}^{M} \alpha_k = \alpha$
β_k	Length/computation space of $Seq2$ processed by P_k, where $\sum_{k=1}^{M} \beta_k = \beta$
ω_k	Inverse of speed of processor P_k (in s/load)
$L_{k,l}$	lth sub-matrix of S, P and Q that is assigned to P_k for computation. All the sub-matrices have the same dimensions for the same value of k
$T(m)$	Processing time, calculated as the time period for the generation of the S, P and Q matrices

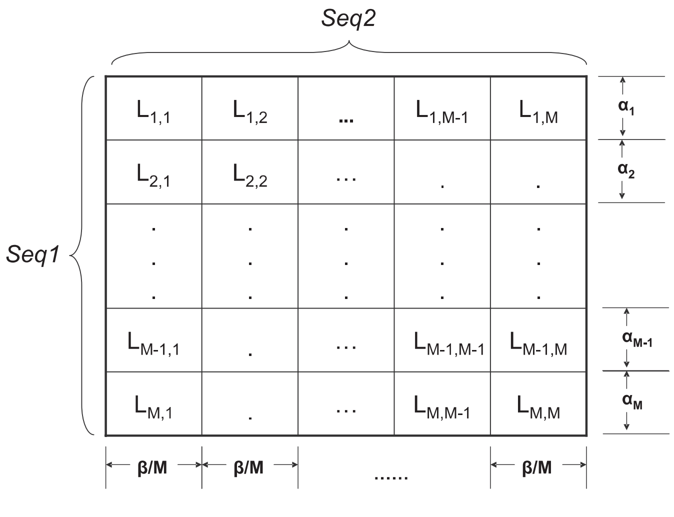

Fig. 4. Distribution of the Smith–Waterman matrices S, P and Q.

3.2. ITM for MSA

In Taylor's strategy, clustering of multiple sequences are done based on pair-wise sequence alignments of the various sequences with each other. A brief description of Taylor's method is presented below and an illustrative example is presented in an Appendix for readers.

After the sequences have been clustered, overlapping alignments are preserved by adding gaps into already pair-wise-aligned sequences to maintain overall cluster alignment. However, this might not assure an improved quality solution or an optimal solution. Therefore, an ITM that is proposed aims to realign sequences before they are added into an order set. Firstly, let X_Y represent modified sequence X due to its alignment with Y. Suppose the ordered set now contains aligned sequences B and D namely, (B_D, D_B). Following the original method, the sequence with the highest pair-wise score with either B or D will be added to the set. Let us suppose that this pair be (A, B). Thus the ordered set will now be (A_B, B_D, D_B). However with ITM, instead of adding A_B to the set, we will now realign A to B_D, thus ending up with a set of (A_{B_D}, B_D, D_B). This is shown to produce a more accurate MSA, as an altered alignment of sequences are considered rather than original sequences.

3.3. Center-to-center heuristic strategy

While individual rows align a subset of sequences, we need to align all these subsets to generate a fully aligned set of sequences. One can follow a simple concatenation of sequences, however, in this case there is no influence or exchange of knowledge of sequences between rows of mesh so as to generate a final alignment that has a better score. In other words, if final alignment involves a comparison among the scores secured by each row, then the quality of the solution can be improved.

We propose a methodology for generating a final alignment using the locally aligned sequences from each row. The method described here will not be manipulating or modifying sequences within the local clusters but rather describing an approach in ordering the clusters. This is also due to the fact that MSA problem usually comprises a very large number of sequences to be aligned thereby increasing the complexity of this sequencing process. Our method, referred to as "center-to-center" strategy (CCS) is proposed to carry out this ordering. In this strategy, the center sequence [3] in all the clusters is extracted into a set and are aligned against each other, similar to a single cluster alignment process. ITM is then used to decide the positioning of these sequences, i.e., ordering is carried out. Once individual positions of these (center) sequences are determined, we consider ordering the individual subsets according to their respective center sequence positions.

The reason why a CCS is proposed as opposed to any other heuristics is due the way ITM (or in fact, Taylor's) operate. In Taylor's methodology, a sequence can be added to the front or to the back of the already aligned set depending on whether it had a highest score affinity with the sequence at the front or at the back of the set. As such, there is no certainty on whether the sequence with the highest similarity score (i.e., the one that is aligned at the very beginning) would end up at the front, back or center. However, the probability of it being in the center of the cluster is higher than being either at the beginning or the end of the cluster. Thus, this CCS is recommended.

4. Processing time calculations

We will describe the methods we have used to compute the serial and parallel processing times. As in the parallel processing literature, there exist several flavors of speed-up [18] and we define our metrics below.

A. Serial time calculation: The serial time, $T(s)$, is defined as the time taken by the fastest processor available on the entire mesh to process all the sequences. This is done to evaluate whether the parallel times are always better compared to serial times, especially when the architecture has fastest serial processors in place for such large scale problems.

B. Parallel time calculation:

DLT approach. As described in the above sections, the multiprocessor strategy essentially parallelizes the computation. However, even with DLT paradigm in place, all the computations cannot proceed simultaneously, due to the dependency in matrix information, as described in Section 2. Only submatrices with the same diagonal values of $(k + l)$ can be computed concurrently. Therefore, the total processing time, referring to Fig. 5 is, $T(p) = t_1 + t_2 + t_3 + t_4 + t_5$.

ELP approach. Here the ELP approach will be used as a benchmark against the DLT approach, to observe any advantage gained in using DLT. In ELP, the computation space of a sequence will be divided equally among all processors, and thus, time taken for each processor to complete their load will vary. Hence, all processes will be limited by the speed of the slowest processor involved. Only when the slowest processor completes its load may the processing in the next computation segment begin. Therefore, the total processing time, referring to Fig. 6 is $T(p) = t_1 + t_2 + t_2' + t_3 + t_3' + t_4 + t_4' + t_5$.

[3] Assume that we have odd number of sequences for the ease of understanding.

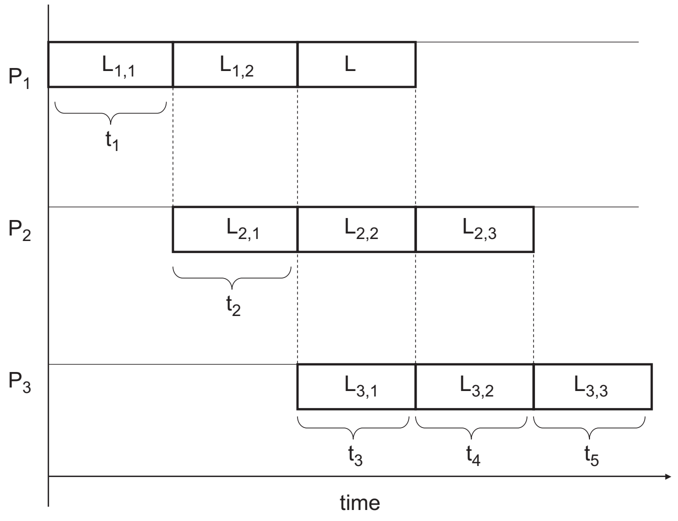

Fig. 5. Parallel time by DLT for a 3-processor problem.

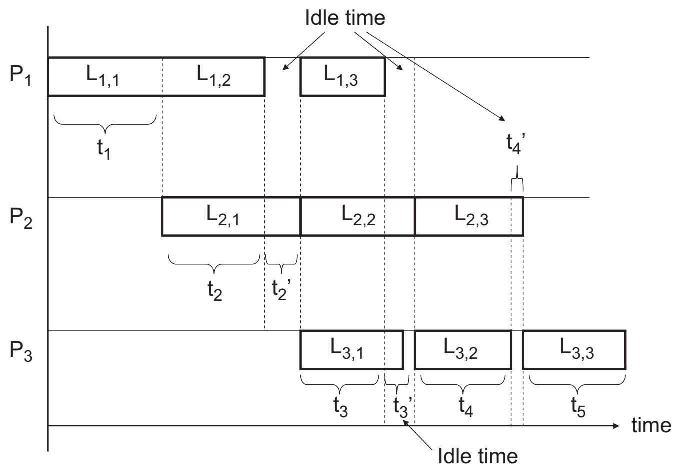

Fig. 6. Parallel time by ELP for a 3-processor problem.

4.1. Definition of speed-up

Before we define the speed-up, we need to define the final parallel processing time. As mentioned in the previous section, the parallel time, $T(p)$ is the time taken by a row to align M number of sequences. As there are N rows, working concurrently with one another, the $T(p)$ defined above is time taken by a particular row, and this may differ from row to row. Therefore, the overall parallel time, taking into consideration N rows, is defined to be

$$\textit{Overall parallel time}, \; T'(p) = \max\{T_i(p)\}, \quad i=1,\ldots,N, \tag{9}$$

where $T_i(p)$ is parallel time obtained from row i.

Thus, speed-up is defined as the ratio of time taken by the fastest processor, serial time, $T(s)$, over the time taken by $N \times M$ processors in the mesh, the parallel time, $T'(p)$:

$$Speed\text{-}up = \frac{T(s)}{T'(p)}. \tag{10}$$

We will use this metric to quantify the performance of the strategies.

5. Performance evaluation

We now describe the environment we used to simulate and conduct performance of our approaches. All the experiments were done utilizing the NUS ACAD-HPC Linux Cluster comprising nodes running on Linux platforms. The 16-node Linux Cluster has 32 2.2 GHz Intel Xeon CPUs for interactive and batch jobs.[4] All experiments are set to run in an automated fashion for every strategy used. Below we describe the settings on the hardware used.

5.1. Simulation settings

In the simulation experiments carried out, the $N \times M$ mesh size was fixed, with N, the number of rows in the mesh, is set to be 10 and M, the number of processors in a row, is set to vary from 2 to 10. The cluster nodes comprises heterogeneous processors, whose speeds are set with one of the values 2, 4, 6, 8 and 10. The sequences used for this simulation are homologous protein sequences derived from the *Rattus Norvegicus* and *Mus Musculus* databases of olfactory receptors and have an average length of approximately 300 amino acid residues each. A total of 200 sequences is used in the serial time, parallel time, and speed-up computations. Sequences used for this simulation have been collected from GenBank [12]. Each simulation run is carried out for 100 times and average results are reported. The SW constants (described in Section 2) used are as in Table 2

We conduct simulation experiments to study two important issues—one on the influence of number of sequences on time and the other on the effect of mesh size (network scalability), use of DLT and ELP techniques. These two experiments serve to analyze the different parameters and factors that affects the parallel time and speed-up of the system in our context. The experiments and results are described in the next two sections.

5.2. Results and discussions

We will present our experiences and results now. We will also discuss on the implications and usefulness of the results.

5.2.1. Effect of mesh size, DLT and ELP on finish time

In this study, we consider 200 protein sequences. The mesh size is varied from a dimension 10×2 to 10×10 to analyze the effect of overall speed-up with respect to the varying number of processors per row. This study is particularly useful when number of sequences to be handled grows very large. Simulation is carried out for both the ELP and DLT techniques. The serial, parallel times and speed-up for both the approaches are recorded. As shown in Fig. 7, the parallel processing times for both the DLT and ELP approach seem to exponentially decrease. The DLT approach always produces a lower finish time than the ELP approach. This is due to the fact that when ELP is used, the faster processors always have to wait for the slowest processor to complete the task in a row (generating longer idle times) due to the computational dependency operations. On the other hand, DLT approach gains an advantage of partitioning the computation space according to a processor's speed to minimize the delay experienced, if at all any exist. Fig. 8 illustrates speed-up performance for both DLT and ELP methods. The speed-up exhibited seems approximately linear in both the cases. The DLT approach, however, delivers a higher speed-up—almost as a function of number of processors per row compared to the ELP approach, again due to the data dependency as described before. Fig. 9 essentially compares the serial and parallel processing times in a row, because each row processes a total of 20 protein sequences, with a total of 200 sequences in the 10 rows of the mesh.

In Fig. 9 it may be observed that the serial time is found to be faster than both parallel approaches at first, however, the DLT approach eventually guarantees better finish times than the serial and parallel ELP strategies. The reason for this behavior initially is due to the heterogeneity (random spread of heterogenous processors) throughout the mesh cluster. For example, with two processors-per-row, there is no guarantee that

Table 2
Smith Waterman parameters

w_k	Gap penalty function	$u + vk$
k	Gap length	Number of gaps inserted into sequence
u	Penalty for initiating gap	-5
v	Penalty for extending gap	-1

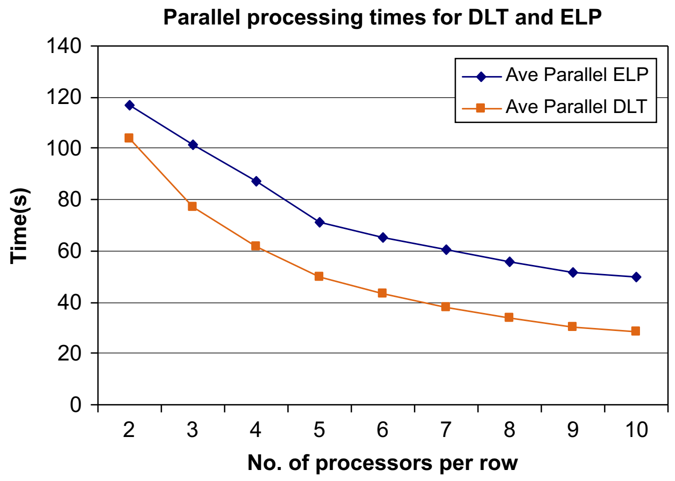

Fig. 7. Parallel processing times of DLT and ELP approach.

[4] ACAD-HPC stands for academic and high performance computing provided by the Supercomputing and Visualization Unit of the Computer Center. Two types of hardware resources are currently accessible through the Portal: a Windows-based PC compute farm and a Linux-based cluster.

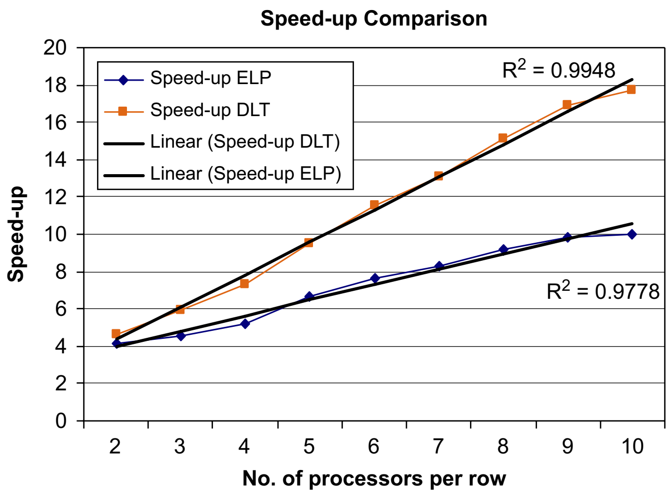

Fig. 8. Speed-up through DLT and ELP approach.

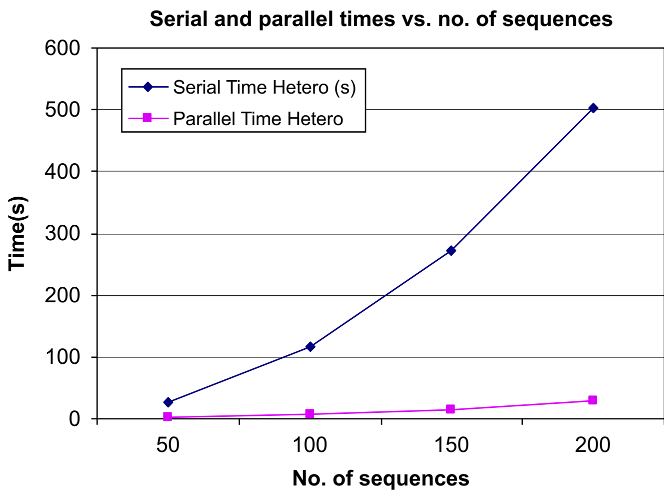

Fig. 10. Serial and parallel time with respect to varying number of sequences.

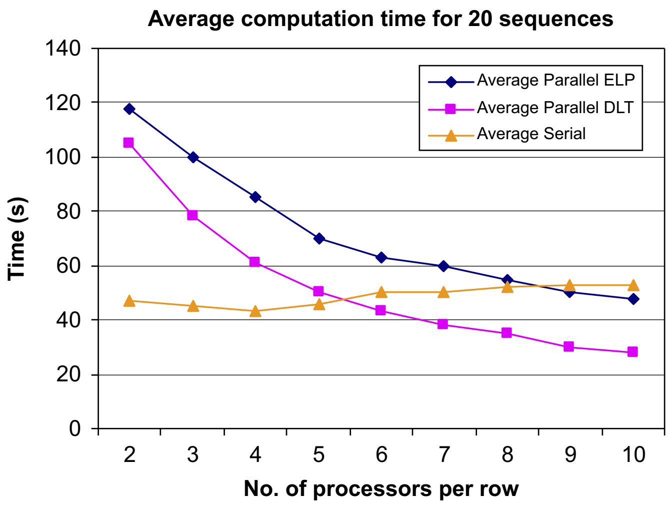

Fig. 9. Average serial and parallel processing times for 20 sequences per row.

Table 3
Number of sequence comparisons

Number of sequences	Number of comparisons
50 (5 per row/cluster)	${}^{5}C_2 \times 10$ *rows* $= 100$
100 (10 per row/cluster)	${}^{10}C_2 \times 10$ *rows* $= 450$
150 (15 per row/cluster)	${}^{15}C_2 \times 10$ *rows* $= 1050$
200 (20 per row/cluster)	${}^{20}C_2 \times 10$ *rows* $= 1900$

their speeds match the fastest processor in the mesh. However, the chances are that they may be slower. Thus, the processing time, although running in parallel could be higher than the serial time. However, increasing the number of processors actually increases the chances of higher and faster computing power, resulting in minimum processing time.

The same argument will not hold for ELP approach, as the processing time always depends on the slowest processor involved. Further, due to network heterogeneity it is very unlikely that the ELP approach will surpass even the serial approach. This is also because the serial time is defined as the time taken by the fastest processor in the mesh. In this set of results (averaged), the ELP reaches the serial approach when 10 processors per row is used.

5.2.2. *Effect of number of sequences on finish time*

In this study, we vary the number of sequences to be processed, while the mesh size is kept constant at 10 × 10 (100 nodes). The number of sequences range from 50 to 200. This is done to analyze the effects on serial and parallel times with respect to the number of sequences. As shown in Fig. 10, both serial and parallel times increases when number of sequences becomes large. It may be noted that the tendency is not strictly linear, but more like an exponential variation. This is essentially due to the fact that the amount of computation (i.e., in this case the number of sequence comparisons and score generation) involved in processing larger set of sequences does not scale up in a linear fashion with increasing number of sequences, as shown in Table 3. Therefore, as the number of comparisons needed increases exponentially, the finish time (both serial and parallel) also found to be increasing exponentially.

5.2.3. *Quality of the solutions*

At this juncture, it is natural to measure the quality of solutions delivered by our approaches. Thus, in our experimentations, we measure the quality by computing the overall score at three different points in our architecture and report our findings. The overall score is defined similar to the case of pair-wise alignment.[5] First set of measures are tapped after Taylor's method (a standard approach), second measure is tapped from the output of ITM module, the third one is the output of our center-to-center heuristic strategy. Following table

[5] Column-wise score computation first and then adding all the column scores for a given alignment.

Table 4
Quality of solutions (averaged over 30 runs)

Number of sequences	S(TM)	S(ITM)	S(C-to-C)
50 (5 per row/cluster)	42	49	53
100 (10 per row/cluster)	63	69	74
150 (15 per row/cluster)	54	60	76
200 (20 per row/cluster)	61	68	117

summarizes the findings wherein S(TM) represents an overall score of multiple sequence alignment after Taylor's method (TM), S(ITM) is an overall score after ITM module, and S(C-to-C) is the refined overall score after our heuristic strategy, respectively. Each entry in the following table is a result of several runs of a single experiment and we took an average score. From the results shown, we observe that the ITM and C-to-C strategy plays a key role in improving the overall cost.

Remark. It should be noted that the results in Table 4 reflects the fact that there is an improvement in quality gradually from TM to ITM and to C-to-C strategy. However, by no means the results must be compared against the number of sequences across each experiment, as the quality of the score depends on net alignment and does not depend on the number of sequences involved.

6. Conclusions and future work

In this paper we have designed a high-performance strategy for mesh-based architectures for processing very large number of biological sequences. Motivated by the fact that current online engines performing MSA impose severe restrictions in handling more than few tens of sequences, our methodology demonstrated here conclusively shows that practically any number of sequences can be handled. We had systematically described the design, and presented the load distribution strategy. We had designed an improved Taylor's methodology and proposed a heuristic (center-to-center) to realign and refine the overall score of the entire set of sequences. With a feedback mechanism in place, we have shown that the mechanism is resilient to the choice of sequences and ultimately evolves an acceptable aligned sequence with a high score. This study is particulary useful to develop online engines that can handle several hundreds of sequences for MSA process. As DLT paradigm is shown to be an invaluable tool in handling large scale data processing for network-based computing systems, we have employed and demonstrated the use of DLT paradigm in our design. For comparative purposes, we have also employed the conventional ELP strategy. Our simulation results generated from clusters clearly produced high quality solutions and DLT approach is shown to generate results in a minimum amount of time. As seen from the approach proposed, it may be observed that although we consider a mesh topology in handling the sequences, an underlying physical infrastructure could be a cluster or any HPC system with a group of compute nodes on which the logical organization could assume a mesh topology. Further, the recursive nature of the problem has been exploited to schedule and the solution approach is readily suitable for SW-like dynamic programming class of algorithms.

Immediate extensions to this work can be in an attempt to practically implement this strategy and to open up a portal for public use. This venture is currently underway. Further, more accurate and comprehensive work can be done on how the processing time is calculated and also the distribution of load to processors in the real-life implementation on a network. Firstly, communication delay time between processors can be factored in to give a more realistic representation of the processing time involved. Secondly, the computational space (load) can be made to be distributed on a wider scale, which means the one can attempt distributing at a much finer level of computation involving both the rows and columns. Although our work uses mesh topology, handling such computationally intensive loads can be easily entertained on computational grids following a software framework that is suitable for parallelizable workloads. One such contribution in the existing literature that suggests a software methodology for grid environments is reported in [14]. The work deals with implementation aspects and also proposes scheduling scheme for a master–slave kind of workable approach. Certainly this is one of the methods that can be used and is applicable for the problem tackled in our paper too. Finally, sophisticated heuristic methods can be designed to improve the overall alignment of sequences. We hope that the contributions in this paper would spur further interest in this direction.

Appendix A. Taylor's method

Below we will illustrate Taylor's clustering strategy as described in [35]. Consider the following table that gives scores for pair-wise alignment of sequences A, B, C, D, and E. Taylor's clustering strategy begins by computing pair-wise alignment scores between all sequences to be clustered. The best score is then identified and placed in the ordered set. From Table 5, this would be (B, D). Then the next best alignment to B or D is identified. This would be (B, A). Thus the sequence A is now enters the ordered set next to its pair, B. The ordered set is now (A, B, D). Next, Taylor's identifies the next best alignment involved with the two end sequences of the ordered set, A and D. The item selected is added to the appropriate end. This continues until the entire set is ordered—(E, A, B, D, C).

Table 5
A pair-wise alignment score matrix

	A	B	C	D	E
A	–				
B	393	–			
C	188	390	–		
D	298	401	372	–	
E	200	254	180	320	

References

[1] S.F. Altschul, W. Gish, W. Miller, E.W. Myers, D. Lipman, A basic local alignment search tool, J. Mol. Biol. 215 (1990) 403–410.

[2] G.J. Barton, M.J. Sternberg, A strategy for the rapid multiple alignment of protein sequences. Confidence levels from tertiary structure comparisons, J. Mol. Biol. 198 (2) (1987) 327–337.

[3] D.A. Benson, K.-M. Ilene, J.L. David, O. James, A.R. Barbara, L.W. David, GenBank, Nucleic Acids Res. 28 (1) (2000) 15–18.

[4] M.P. Berger, P.J. Munson, A novel randomized iteration strategy for aligning multiple protein sequences, Comput. Appl. Biosci. 7 (1991) 479–484.

[5] V. Bharadwaj, G. Debasish, G.R. Thomas, Divisible Load Theory: A New Paradigm for Load Scheduling in Distributed Systems, Special Issue on Divisible Load Scheduling in Cluster Computing, vol. 6(1), Kluwer Academic Publishers, Dordrecht, 2003.

[6] V. Bharadwaj, G. Debasish, M. Venkataraman, G.R. Thomas, Scheduling Divisible Loads in Parallel and Distributed Systems, IEEE Computer Society Press, Los Almitos, California, 1996.

[7] A. Califano, I. Rigoutsos, FLASH: a fast look-up algorithm for string homology, in: Proceedings of the First International Conference on Intelligent Systems for Molecular Biology, 1993, pp. 56–64.

[8] CLUSTALW, ⟨http://www.ebi.ac.uk/clustalw/⟩.

[9] DNA Data Bank of Japan, ⟨http://www.ddbj.nig.ac.jp⟩.

[10] I. Eidhammer, I. Jonassen, W.R. Taylor, Protein Bioinformatics: An Algorithmic Approach to Sequence and Structure Analysis, Wiley, New York, 2004 ISBN: 0-470-84839-1.

[11] W. Feng, Green Density + mpiBLAST = Bioinfomagic, in: The Proceedings of International Conference on Parallel Computing (ParCo 2003), September 2003.

[12] GenBank, ⟨http://www.ncbi.nlm.nih.gov⟩.

[13] O. Gotoh, An improved algorithm for matching biological sequences, J. Mol. Biol. 162 (1982) 705–708.

[14] J.P. Goux, S. Kulkarni, J. Linderoth, M. Yoder, An enabling framework for master–worker applications on the computational grid, in: Proceedings of High Performance Distributed Computing (HPDC), 2000, pp. 43–50.

[15] K. Ko, T.G. Robertazzi, Equal allocation scheduling for data intensive applications, IEEE Trans. Aerospace Electron. Systems (2004).

[16] D.J. Lipman, W.R. Pearson, Rapid and sensitive protein similarity searches, Science 227 (1985) 1435–1441.

[17] H.M. Martinez, A flexible multiple sequence alignment program, Nucleic Acids Res. 16 (5 Part A) (1988) 1683–1691.

[18] J.Q. Michael, Parallel Programming in C with MPI and OpenMP, McGraw-Hill, New York, USA, 2003.

[19] E.W. Myers, An O(ND) difference algorithm and its variations, Algorithmica 1 (2) (1986) 251–266.

[20] S.B. Needleman, C.D. Wunsch, A general method applicable to the search for similarities in the amino acid sequence of two sequences, J. Mol. Biol. 48 (1970) 443–453.

[21] W.R. Pearson, Rapid and sensitive sequence comparison with FASTA and FASTP, Methods Enzymology 183 (1990) 63–98.

[22] W.R. Pearson, D.J. Lipman, Improved tools for biological sequence comparison, Proc. Nat. Acad. Sci. USA 85 (1988) 2444–2448.

[23] D. Pekurovsky, I.N. Shindyalov, P.E. Bourne, A case study of high-throughput biological data processing on parallel platforms, Bioinformatics 20 (2004) 1940–1947.

[24] T.G. Robertazzi, Ten reasons to use divisible load theory, Computer 36 (5) (2003) 63–68.

[25] T. Rognes, S. Erling, Six-fold speed-up of Smith Waterman sequence database searches using parallel processing on common microprocessors, Bioinformatics 16 (8) (2000) 699–706.

[26] P.H. Sellers, On the theory and computation of evolutionary distances, SIAM J. Appl. Math. 26 (1974) 787–793.

[27] T.F. Smith, M.S. Waterman, Identification of common molecular subsequence, J. Mol. Biol. 147 (1981) 195–197.

[28] W.R. Taylor, Multiple sequence alignment by a pairwise algorithm, Comput. Appl. Biosci. 3 (2) (1987) 81–87.

[29] The EMBL (European Molecular Biology Laboratory) Nucleotide Sequence Database, ⟨http://www.ebi.ac.uk/embl⟩.

[31] O. Trelles, M.A. Andrade, A. Valencia, E.L. Zapata, J.M. Carazo, Computational space reduction and parallelization of a new clustering approach for large groups of sequences, Bioinformatics 14 (5) (1998) 439–451.

[32] M.S. Waterman, Mathematical Methods for DNA Sequences, CRC Press Inc., Boca Raton, FL, 1986.

[33] M.S. Waterman, T.F. Smith, W.A. Beyer, Some biological sequence metrics, Adv. Math. 20 (1976) 367–387.

[34] H.M. Wong, V. Bharadwaj, Aligning biological sequences on distributed bus networks: a divisible load scheduling approach, IEEE Trans. Inform. Technol. Biomed. 9 (4) (2005) 1910–1924.

[35] T.K. Yap, O. Frieder, R.L. Martino, High Performance Computational Methods for Biological Sequence Analysis, Kluwer Academic Publishers, Dordrecht, 1996.

[36] T.K. Yap, O. Frieder, R.L. Martino, Parallel computation in biological sequence analysis, IEEE Trans. Parallel Distrib. Systems 9 (3) (1998).

[37] X. Zhang, Y. Yan, Modeling and characterizing parallel computing performance on heterogeneous networks of workstations, in: Proceedings of the Seventh IEEE Symposium on Parallel and Distributed Processing, October 1995.

Diana H.P. Low is currently a graduate student in Computational and Systems Biology under the Singapore–MIT Alliance Graduate Fellowship—a collaboration between The National University of Singapore and Massachusetts Institute of Technology. An ASEAN scholar, she received her B.Eng. in Electrical Engineering from The National University of Singapore in 2005. Her research interests includes molecular immunology, computational biology and biological engineering.

Bharadwaj Veeravalli, Senior Member, IEEE & IEEE-CS, received his Ph.D. from Department of Aerospace Engineering, Indian Institute of Science (IISc), Bangalore, India in 1994, Master's in Electrical Communication Engineering from IISc, Bangalore, India in 1991 and B.Sc in Physics, from Madurai-Kamaraj University, India in 1987. He did his post-doctoral research in the Department of Computer Science, Concordia University, Montreal, Canada, in 1996. He is currently with the Department of Electrical and Computer Engineering, Communications and Information Engineering Division, at The National University of Singapore, Singapore, as a tenured Associate Professor. His main stream research interests include, Cluster/Grid computing, Scheduling in Parallel and Distributed systems, Bioinformatics, Multiprocessor systems, and Multimedia computing. He has published extensively in high-quality international journals and conferences and has co-authored several book chapters and had published three research monographs in the areas of Parallel and Distributed Systems, Distributed Databases, and Networked Multimedia Systems, in the years 1996, 2003 and 2005, respectively. He is currently serving the Editorial Board of IEEE Transactions on Computers, IEEE Transactions on SMC-A, and International Journal of Computers and Applications, USA, as an Associate Editor.

David A. Bader is an Associate Professor in Computational Science and Engineering, a division within the College of Computing, Georgia Institute of Technology. He received his Ph.D. in 1996 from The University of Maryland, was awarded a National Science Foundation (NSF) Postdoctoral Research Associateship in Experimental Computer Science. He is an NSF CAREER Award recipient, an investigator on several NSF awards, and supported by research awards from IBM, Sony, Microsoft Research and Sun Microsystems. Dr. Bader serves on the Steering Committees of the IPDPS and HiPC conferences, and was the General co-Chair for IPDPS (2004–2005), and Vice General Chair for HiPC (2002–2004). David has chaired several major conference program committees: Program Chair for HiPC 2005, Program Vice-Chair for IPDPS 2006 and Program Vice-Chair for ICPP 2006. He has served on numerous conference program committees related to parallel processing and computational science and engineering, is an associate editor for several publications including the IEEE Transactions on Parallel and Distributed Systems (TPDS), the ACM Journal of Experimental Algorithmics (JEA), IEEE DSOnline, and Parallel Computing, is a Senior Member of the IEEE Computer Society and a Member of the ACM. Dr. Bader has been a pioneer in the field of high-performance computing for problems in bioinformatics and computational genomics. He has co-chaired a series of meetings, the IEEE International Workshop on High-Performance Computational Biology (HiCOMB), written several book chapters, and co-edited special issues of the Journal of Parallel and Distributed Computing (JPDC) and IEEE TPDS on high-performance computational biology. He has co-authored over 80 articles in peer-reviewed journals and conferences, and his main areas of research are in parallel algorithms, combinatorial optimization, and computational biology and genomics.