Finding Fractional Identities in Multiple Sequences Using a Fast Parallel Algorithm

Evandro A. Marucci[1], Geraldo F. D. Zafalon[1,2], Aleardo Manacero[1], Liria M. Sato[2], José M. Machado[1]

[1]Departamento de Ciências de Computação e Estatística
Universidade Estadual Paulista - UNESP

[2]Departamento de Engenharia de Computação e Sistemas Digitais
Escola Politécnica - Universidade de São Paulo - USP

{marucci,zafalon,liriasato}@gmail.com, {aleardo,jmarcio}@ibilce.unesp.br

Abstract. *The calculation of similarities is an operation widely used in Bioinformatics. Usually made between pairs of sequences, it determines the relative amount of common residues between them. The fractional identity is a measure of similarity used in many sequence alignment tools. Due to the growing of volume of comparative data, the interest in parallel computing is increasing remarkably. The implementation of a parallel algorithm for finding fractional identities in multiple sequences is proposed in this paper. Tests have shown that the algorithm presents a very good scalability and a nearly linear speedup.*

1. Introduction

The use of sequence comparison methods has been growing remarkably in recent years, in response to the volume increasing of available genomic data. Consequently, the development of new approaches that reduce the processing time are fundamental to the improvement of this area.

In general, the sequences comparison starts with the search of similarity degree between pairs of sequences. This search is made by methods which may require or not that the sequences are aligned. A method widely used by the most of sequence alignment tools and which requires a prior alignment is the fractional identity [Edgar 2004a] method. CLUSTALW [Thompson et al. 1994] and MUSCLE[Edgar 2004b] use this method in one of their steps.

We propose in this paper a parallel algorithm for fractional identity method and show implementation information [1]. This algorithm can be used in the development of any parallel tool which needs a stage of fast refinement with a great biological accuracy.

2. Fractional Identity

Given an alignment, we ignore all positions with gaps and, in the other positions, we calculate the amount of them with equal residues in relation to the total amount of positions

[1]This work was partially supported by the São Paulo Research Foundation (FAPESP - Brazil) under Grant No. 06/59592-0
Corresponding author: Geraldo F.D. Zafalon

with any kind of residues. Figure 1 gives an example of the fractional identity calculation. In this figure, the aligned pair of sequences has 9 positions. Among them, only positions 1, 2, 4, 5, 6, 7, 9 and 10 have no gaps. Positions 2, 4, 6, 7 and 9, in turn, are those having equal residues in both sequences. Thus, 8 positions without gaps and 5 positions with equal residues. The fractional identity between these sequences is therefore $5/8$.

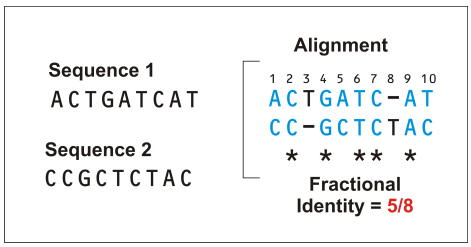

Figure 1. *Fractional identity calculation between two sequences.*

3. Parallel Algorithm

This algorithm dynamically divides the computation among the processors through a master-slave approach. This parallelism is performed by distributing the similarity calculation of sequence pairs among the available slaves. This is possible because each fixed pair similarity calculation is independent from the remaining pair similarity calculations. The flowchart of this algorithm is shown in figure 2.

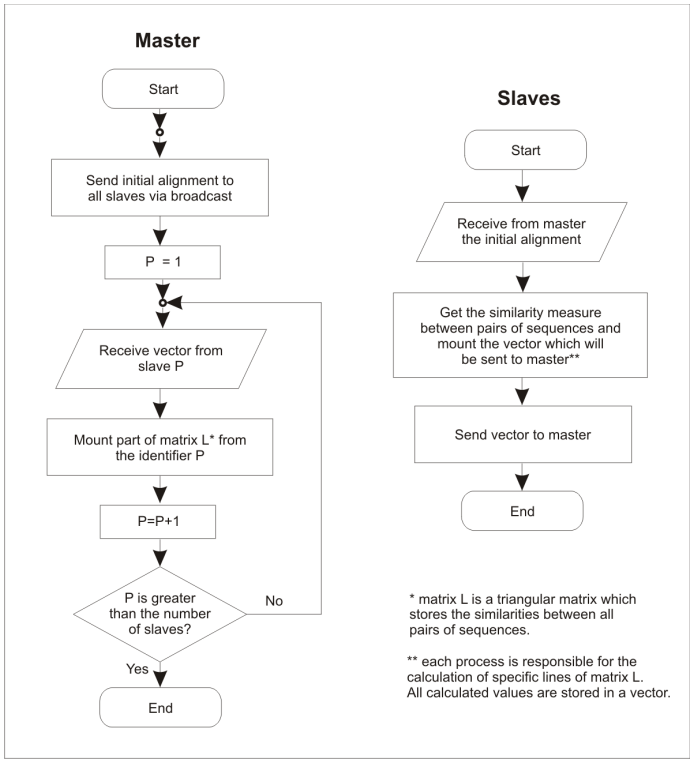

Figure 2. *Parallel algorithm flowchart for the calculation of fractional identities in multiple sequences.*

Initially, the master sends all sequences by broadcast to all slaves. The task distribution is based on the processor identifier, assigning to each slave the calculation of

specific lines in the similarity triangular matrix. This matrix is obtained as exemplified in figure 3. Each slave initially calculates the line corresponding to the identifier, in a p-step loop, where p is the number of processors. In the example, we have eight sequences, and, hence, a seven by seven matrix. The first slave is responsible for the calculation of the lines from sequences 1, 4 and 7. The second one by lines of sequences 2 and 5 and the third by lines of sequences 3 and 6. The gray values are the results obtained in other slaves; they will be joined in the master for the creation of the similarity matrix. Although simple, this scheduling tends to a good load balancing for any number of sequences involved.

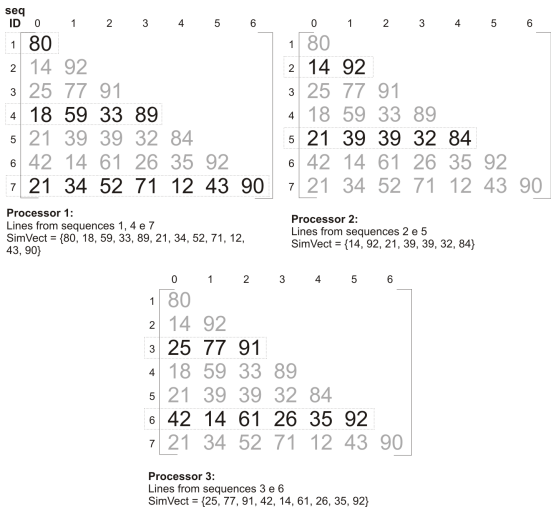

Figure 3. *Example of how the calculation of the similarity matrix is distributed between the slaves.*

4. Tests and Results

In order to verify the proposed algorithm performance, we executed tests with four different datasets. For each dataset, we verified the algorithm scalability when executed in a growing number of machines. All tests were performed on a Beowulf cluster.

Some executed tests are showed in the figure 4. There are four graphics listed from 1 to 4. Basically, in this figure are showed some graphics which have in the horizontal axis the number of processors and in the vertical axis the running time in seconds. On the top of each graphic are placed some parameters of the sequence datasets used in the tests. The parameter *num* represents the number of sequences used in the execution, the parameter *max* represents the maximum number of residues found among the sequences of dataset and the parameter *avg* shows the average number of residues found among the sequences of dataset.

The first three graphics illustrated in figure 4 show an almost linear speedup for tests with more than 500 sequences. The first one with 567 sequences, the second with 988 sequences and the third one with 2000 sequences.

The last graphic (number 4) shows a constant performance, regardless of the number of machines, for an entry with few sequences. In this case, the sequential implementation of the algorithm is so fast for that input (approximately 1 second) that the speedup achieved with tasks division is close to the time spent with message exchanges.

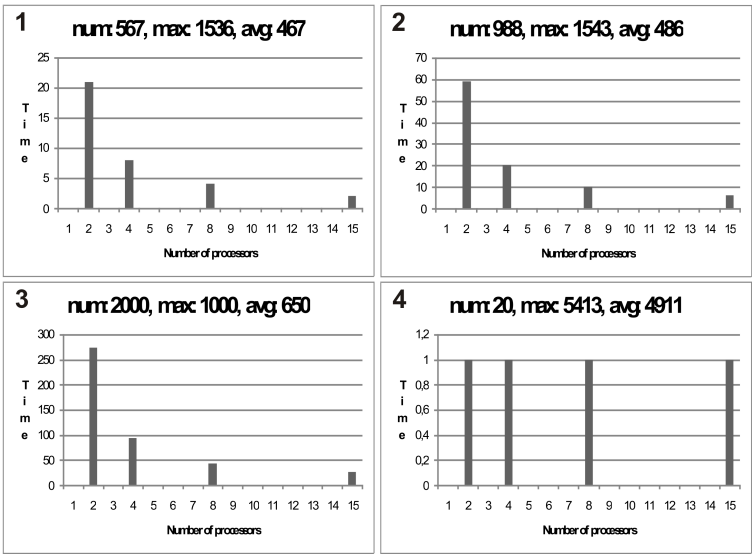

Figure 4. *Running time of the parallel algorithm for 2, 4, 8 and 15 processors with four different datasets.*

The performance achieved with the addition of 14 processors is more than 10 times higher. In tests we can realize a nearly linear speedup independent from the dataset used and the number of processors in the cluster.

5. Conclusions

In this article, we propose and measure a parallel strategy to calculate similarities between multiple sequence pairs by using fractional identities. This calculation is used, for instance, in multiple sequence alignment tools. The proposed parallel algorithm has been implemented for distributed memory systems, due to the wide use of Beowulf clusters in genomic research.

The performed tests show that the algorithm presents a good scalability and a nearly linear speedup. With the use of 14 processing nodes (slaves), the system achieved a 10 times more speedup. This speedup is justified by the total independence of the scheduled tasks and by the good load balancing obtained with the merge distribution of triangular matrix lines. Additionally, the communication cost is minimized because the processed data in slaves are sent to the master in a single message.

References

Edgar, R. C. (2004a). Local homology recognition and distance measures in linear time using compressed amino acid alphabets. *Nucleic Acids Research*, 32(1):380–385.

Edgar, R. C. (2004b). Muscle: a multiple sequence alignment method with reduced time and space complexity. *BMC Bioinformatics*, 5(1).

Thompson, J. D., Higgins, D. G., and Gibson, T. J. (1994). Clustal w: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice. *Nucleic Acids Res*, 22(22):4673–4680.