

Comparison of Genomes using High-Performance Parallel Computing

N. F. Almeida Jr*
Univ. Federal de Mato Grosso do Sul
nalvo@dct.ufms.br

E. N. Caceres†
Univ. Federal de Mato Grosso do Sul
edson@dct.ufms.br

C. E. R. Alves
Universidade São Judas Tadeu
prof.carlos_r_alves@usjt.br

S. W. Song‡
Universidade de São Paulo
song@ime.usp.br

Abstract

*Comparison of the DNA sequences and genes of two genomes can be useful to investigate the common functionalities of the corresponding organisms and get a better understanding of how the genes or groups of genes are organized and involved in several functions. In this paper we use high-performance parallel computing to compare the whole genomes of two organisms, namely *Xanthomonas axonopodis* pv. *citri* and *Xanthomonas campestris* pv. *campestris*, each with more than five million base-pairs. Our purpose is two-fold. First we intend to exploit the high-performance power of a cluster of low-cost microcomputers, propose a parallel solution to this problem, and show its feasibility with implementation and performance results. Second we do additional comparisons of the two genomes by locating and compare not only the homologous genes (expressed in terms of the 20-letter amino acids) but also compare the regions or gaps (in terms of the 4-letter DNA nucleotides) between the corresponding homologous genes. We have implemented the proposed comparison strategy to compare the two genomes *Xanthomonas axonopodis* pv. *citri* (*Xac*) and *Xanthomonas campestris* pv. *campestris* (*Xcc*). The parallel platform used is a Beowulf cluster of 64 nodes consisting of low cost microcomputers. *Xac* has 5,175,554 base pairs and 4,313 protein-coding genes while *Xcc* has 5,076,187 base pairs and 4,182 protein-coding genes. The parallel solution is based on the dynamic programming approach and presents not only less processing time, but also better quality results as compared to approaches based on Blast and EGG.*

* Partially supported by CNPq.

† Partially supported by FINEP-PRONEX-SAI Proc. No. 76.97.1022.00, CNPq, FAPESP Proc. No. 1997/10982-0.

‡ Partially supported by CNPq Grants No. 52.3778/96-1, 55.2028/02-9.

1. Introduction

High-performance computing has always been associated with solutions of the so-called *grand challenges* that include such diverse areas as weather prediction, climate and global changes, material sciences, medicine and health, and more recently, the important area of bioinformatics. With recent advances of DNA sequencing technology, whole genomes of bacteria, archaea and eukaryota, including humans, have been sequenced, with the announcements of new sequencing results on a nearly monthly basis. The amount of genome information already available is astounding. According to the National Center for Biotechnology Information of the National Institute of Health, within 20 years after the creation of Genbank, the database has grown from 680 thousand base pairs in 1982 to 28.5 billion base pairs in 2002 [12].

In an area so overwhelmed by data and incomplete information we are urgently in need of high-performance computing tools to analyze and interpret the vast amount of raw data and fragments of information in order to get a better glimpse and understanding of at least some of the biological functions of living organisms.

Though an awesome and computationally expensive effort, comparison of the DNA sequences and genes of two whole genomes can be useful to investigate the common functionalities of the corresponding organisms and get a better understanding of how the genes or groups of genes are organized and involved in several functions.

In this paper we use high-performance parallel computing to compare the whole genomes of two organisms, namely *Xanthomonas axonopodis* pv. *citri* and *Xanthomonas campestris* pv. *campestris*, each with more than five million base-pairs.

A pioneer work along the line of comparison of these two genomes has appeared in the literature, in a paper coauthored by Rasera da Silva et al. [13]. One of the approaches used in [13] consists of locating the homologous genes (ex-

pressed in terms of amino acids) of the two genomes and compare each pair of the homologous genes.

Our purpose is two-fold. First we intend to exploit the high-performance power of a cluster of low-cost microcomputers, propose a parallel solution to this problem, and show its feasibility with implementation and performance results. Second we intend to do additional comparisons of the two genomes not handled by the previous work. More specifically, we locate and compare not only the homologous genes (expressed in terms of the 20-letter amino acids) but also compare the regions or gaps (in terms of the 4-letter DNA nucleotides) between the corresponding homologous genes.

In addition to [13], comparison of whole genomes has been investigated by Setubal and Almeida [15] and by some related works presented in a special workshop dedicated especially to this purpose [9]. Other works related to the study of genomes, e.g. *Agrobacterium tumefaciens*, include the papers by Wood et al. [18, 19].

2. Basic Concepts

With the intent of making this text readable to those not familiar with terms of Biology, though aware of the risk of oversimplification, we now present some of the basic concepts. The interested reader can obtain further explanation in some excellent texts such as [10, 16].

DNA (deoxyribonucleic acid) is the chemical material in a cell that carries the genetic codes for living organisms. Its structure is a double helix consisting of two sequences of letters from a four-letter alphabet (A, T, C, G), such that A is paired with T, and C with G. The letters represent the *nucleotides* or *bases* known as *adenine*, *thymine*, *cytosine* and *guanine*. Since the bases are paired, they are referred to as *base pairs*.

All the DNA of a living organism is called its *genome*. The size of a genome can vary from millions of base pairs for bacteria to several billions base pairs in the case of mammals. For example, the genomes of the bacteria *Xanthomonas axo-no-po-dis* pv. *citri* (in short *Xac*) and *Xanthomonas campestris* pv. *campestris* (in short *Xcc*) have, respectively, 5,175,554 and 5,076,187 base pairs [13].

Genes are pieces of DNA and most contain the code for making a specific protein. *Proteins* are sequences of amino acids from a 20-letter alphabet. Each amino acid is coded by three nucleotides called a *codon*. There are therefore $4^3 = 64$ possible codon values. However, some different codons code the same amino acid while other codons refer to the start and end of the sequence of amino acids of a proteins. Thus there are 20 different amino acids (and not 64).

3. Comparison Strategy

Consider two genomes of close evolutionary relationship. The term *homology* is used to indicate the fact that two genes share a common evolutionary past. Two genes are *homologous* if they are descendants of a same ancestral gene. Often the similarity between two sequences of DNA or amino acids is important because through similarity we may infer homology. This is because closely related organisms often present similar sequences. Homology in turn is important because often homology determines function. Thus we have the chain *similarity* \rightarrow *homology* \rightarrow *function*.

The comparison of genomes has the potential of revealing evolutionary relationships of seemingly different organisms. *Xanthomonas* for example is a group of bacteria, where *Xanthomonas axonopodis* pv. *citri* or *Xac* causes citrus canker in citrus trees, while *Xanthomonas campestris* pv. *campestris* or *Xcc* causes black rot in crucifers. These two seemingly different bacteria are closely related and it has been shown that *Xac* and *Xcc* share more than 80% of the genes [13].

In this section we summarize a comparison strategy to allow us to identify common regions in two closely related genomes. The output is a global alignment of both DNA sequences that takes into account the homologous genes found between the two genomes. It is crucial to point out that we do not want to obtain merely brute similarity between two genomes, as is possible by using MUMmer [17], a tool for aligning whole genome sequences. Instead we wish to get an alignment of the genomes that takes into consideration biological or functional information.

We use the strategy of splitting the genomes into contiguous blocks and align each pair of blocks, one from each genome. This is better explained as follows.

We use the routines *DP-score*, *DP-align* and *LCS* to get the alignment score of two sequences by *dynamic programming*, to get the alignment proper of two sequences by *dynamic programming*, and to get the *longest common subsequence* of two sequences, respectively. These routines will be explained later.

Consider genomes *G* and *H*. Algorithm 1 summarizes the comparison strategy. Refer also to Fig. 1 and Fig. 2 that illustrate some of the steps.

Fig. 1 shows an example after steps 1 through 3, with the homologous pairs determined and labeled. Fig. 2 shows the situation after step 4. It also shows the intergenic regions $[g, g']$ and $[h, h']$.

4. Parallel String Similarity

We have presented some parallel algorithms for finding the similarity between two strings [4, 5, 6]. Among these we choose the one that is very efficient in practice [6]. It

Algorithm 1 Comparison of Genomes G and H

Input: (i) the 4-letter alphabet DNA sequence of each genome G and H ; (ii) the annotation with the start and end positions of the predicted genes in each DNA sequence; (iii) the genes given in terms of amino acids from a 20-letter alphabet.

Output: Alignment of the genomes G and H .

- 1: *Find pairs of the homologous genes:*
 For all g of G , obtain h of H such that

$$\text{DP-score}(g, h) = \max \{ \text{DP-score}(g, w) \text{ for all } w \text{ of } H \}$$
 - 2: *Label the homologous genes of G :*
 Label the homologous genes of G as $1, 2, \dots, m$ in the same order as their positions in the genome G .
 Let $\text{Label}G$ denote the sequence of labels obtained in this step.
 - 3: *Label the corresponding homologous genes of H :*
 For all pairs of homologous genes (g, h) , g of G , h of H , label gene h with the same label of g .
 Let $\text{Label}H$ denote the sequence of labels obtained in this step.
 - 4: *Find the non-crossing pairs of homologous genes:*
 Obtain the $\text{LCS}(\text{Label}G, \text{Label}H)$. the LCS obtained contains only the non-crossing pairs
 - 5: *Align each pair of homologous genes:*
 For each non crossing homologous pair (g, h) do $\text{DP-align}(g, h)$.
 - 6: *Align each pair of intergenic regions:*
 For each intergenic region $[g, g']$, where g, g' of G are two consecutive genes of the LCS , obtain the corresponding intergenic region $[h, h']$ in H and do $\text{DP-align}([g, g'], [h, h'])$.
 - 7: *Join all the alignments:*
 Concatenate the alignments of the homologous genes and the intergenic regions, in the same order they appear in the genomes.
-

is a parallel wavefront algorithm for efficient biological sequence comparison using the dynamic programming technique. We summarize the algorithm in the section. Details can be found in [6].

This parallel algorithm uses the dynamic programming technique to compute an alignment between two strings A and C , with $|A| = m$ and $|C| = n$. On a distributed memory parallel computer of p processors each with $O((m+n)/p)$ memory, the proposed algorithm requires $O(p)$ communication rounds and $O(mn/p)$ local computing time. A nice feature of this algorithm is that it based on a compromise between the workload of each processor and the number of communication rounds required, expressed by a parameter called α . The algorithm is expressed in terms of this parameter that can be tuned to obtain the best overall parallel

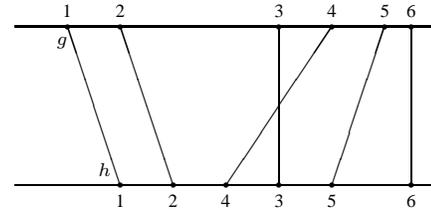


Figure 1. Homologous pairs and their labeling

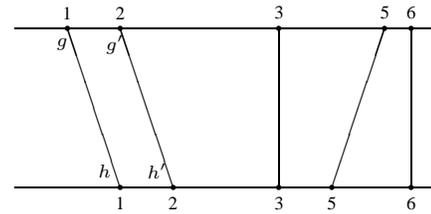


Figure 2. Non crossing homologous pairs and corresponding intergenic regions (e.g. $[g, g']$ and $[h, h']$)

time in a given implementation. Very promising experimental results have been obtained in experimental executions. A characteristic of the wavefront communication requirement is that each processor communicates with few other processors. This makes it very suitable as a potential application for cluster and grid computing.

Let us recall the similarity problem we wish to solve. Let $A = a_1 a_2 \dots a_m$ and $C = c_1 c_2 \dots c_n$ be two strings over some alphabet I . To align the two strings, we insert spaces in the two sequences in such way that they become equal in length. See Fig. 3 and Fig. 4 where each column consists of a symbol of A (or a space) and a symbol of C (or a space). An *alignment* between A and C is a matching of the symbols $a \in A$ and $c \in C$ in such way that if we draw lines between the matched symbols, these lines cannot cross each other. The alignment shows the similarities between the two strings. In the two simple alignment exam-

A	a	c	t	t	c	a	-	t	3
C	a	t	t	c	-	a	c	g	
Score	1	0	1	0	0	1	0	0	

Figure 3. Alignment Example (a).

A	a	c	t	t	c	a	-	t	
C	a	-	t	t	c	a	c	g	
Score	1	0	1	1	1	1	0	0	5

Figure 4. Alignment Example (b).

ples we assign a score of 1 when the aligned symbols in a column match and 0 otherwise. The alignment of Fig. 3 has a score of 5 and than that of Fig. 4 a score of 3.

A more general score assignment for a given alignment between strings is done as follows. Each column of the alignment receives a certain value depending on its contents and the total score for the alignment is the sum of the values assigned to its columns. Consider a column consisting of symbols r and s . If $r = s$ (i.e. a *match*), it will receive a value $p(r, s) > 0$. If $r \neq s$ (a *mismatch*), the column will receive a value $p(r, s) < 0$. Finally, a column with a space in it receives a value $-k$, where $k \in \mathbb{N}$. We look for the alignment (*optimal alignment*) that gives the maximum score. This maximum score is called the *similarity* between the two strings to be denoted by $sim(A, C)$ for strings A and C . There may be more than one alignment with maximum score [16].

Consider $|A| = m$ and $|C| = n$. We can obtain the solution by computing all the similarities between arbitrary prefixes of the two strings starting with the shorter prefixes and use previously computed results to solve the problem for larger prefixes. There are $m + 1$ possible prefixes of A and $n + 1$ prefixes of C . Thus, we can arrange our calculations in an $(m + 1) \times (n + 1)$ matrix S where each $S(r, s)$ represents the similarity between $A[1..r]$ and $C[1..s]$, that denote the prefixes $a_1 a_2 \dots a_r$ and $c_1 c_2 \dots c_s$, respectively.

The similarity score S of the alignment between strings A and C can be computed as follows:

$$S(r, s) = \max \begin{cases} S[r, s - 1] - k \\ S[r - 1, s - 1] + p(r, s) \\ S[r - 1, s] - k \end{cases}$$

Observe that we can compute the values of $S(r, s)$ by using the three previous values $S(r - 1, s)$, $S(r - 1, s - 1)$ and $S(r, s - 1)$, because there are only three ways of computing an alignment between $A[1..r]$ and $C[1..s]$. We can align $A[1..r]$ with $C[1..s - 1]$ and match a space with $C[s]$, or align $A[1..r - 1]$ with $C[1..s - 1]$ and match $A[r]$ with $B[s]$, or align $A[1..r - 1]$ with $C[1..s]$ and match a space with $A[r]$.

An $l_1 \times l_2$ *grid DAG* (Fig. 5) is a directed acyclic graph whose vertices are the $l_1 l_2$ points of an $l_1 \times l_2$ grid, with edges from grid point $G(i, j)$ to the grid points $G(i + 1, j)$ and $G(i + 1, j + 1)$. It is easy to see that the similarity problem can be viewed as computing the minimum

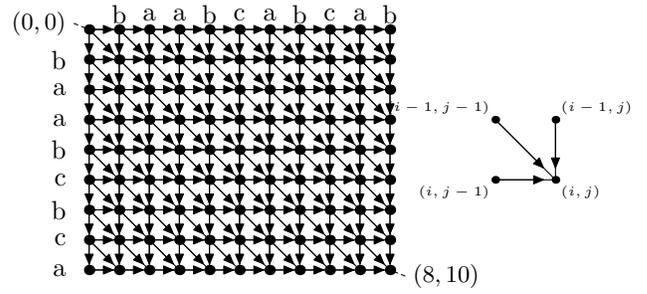


Figure 5. Grid DAG G for $A = \text{baabcba}$ and $B = \text{baabcab}$.

source-sink path in a grid DAG. In Fig. 5 the problem is to find the minimum path from $(0,0)$ to $(8,10)$.

We use the CGM (coarse-grained multicomputer) model with p processors, where each processor has $O(mn/p)$ local memory. It can be seen that this can be reduced to $O((m + n)/p)$.

To compute the similarity matrix S by p processors, the string A is broadcasted to all processors, and the string C is divided into p pieces, of size $\frac{n}{p}$, and each processor P_i , $1 \leq i \leq p$, receives the i -th piece of C ($c_{(i-1)\frac{n}{p}+1} \dots c_{i\frac{n}{p}}$). Algorithm 2 presents the details. The notation P_i^k denotes the work of Processor P_i at round k . Thus initially P_1 starts computing at round 0. Then P_1 and P_2 can work at round 1, P_1 , P_2 and P_3 at round 2, and so on. In other words, after computing the k -th part of the sub-matrix S_i (denoted S_i^k), processor P_i sends to processor P_{i+1} the elements of the right boundary (rightmost column) of S_i^k . These elements are denoted by R_i^k . It is easy to see that in round k , all processors P_i work, where $1 \leq i \leq k$. Since the total number of rounds is increased with smaller values of α the processors start working earlier.

It can be proven that Algorithm 2 uses $(1 + 1/\alpha)p - 2$ communication rounds with $O(\frac{mn}{p})$ sequential computing time in each processor [6].

The routine $\text{DP-align}(x, y)$ is similar to $\text{DP-score}(x, y)$. In addition to obtaining the score of the alignment, it also obtains the alignment proper.

5. Parallel Longest Common Subsequence

Given two strings X and Y of lengths m and n , respectively, the *longest common subsequence* (LCS) problem obtains the length of the subsequences common to X and Y .

Consider a string of symbols from a finite alphabet. A *substring* of a string is any contiguous fragment of the given string. A *subsequence* of a string is obtained by deleting zero or more symbols from the original string. A subsequence can thus have noncontiguous symbols of a string.

Algorithm 2 DP-score: Parallel Similarity Algorithm

Input: (1) The number p of processors; (2) The number i of the processor, where $1 \leq i \leq p$; and (3) The string A and the substring C_i of size m and $\frac{m}{p}$, respectively; (4) The constant α .

Output: $S(r, s) = \max\{S[r, s-1] - k, S[r-1, s-1] + p(r, s), S[r-1, s] - k\}$, where $(i-1)\frac{m}{\sqrt{p}} + 1 \leq r \leq i\frac{m}{\sqrt{p}}$ and $(j-1)\frac{n}{p} + 1 \leq s \leq j\frac{n}{p}$.

- (1) **for** $1 \leq k \leq \frac{p}{\alpha}$
 (1.1) **if** $i = 1$ **then**
 (1.1.1) **for** $\alpha(k-1)\frac{m}{p} + 1 \leq r \leq \alpha k\frac{m}{p}$ **and**
 $1 \leq s \leq \frac{n}{p}$
 compute $S(r, s)$;
 (1.1.2) send (R_i^k, P_{i+1}) ;
 (1.2) **if** $i \neq 1$ **then**
 (1.2.1) receive (R_{i-1}^k, P_{i-1}) ;
 (1.2.2) **for** $\alpha(k-1)\frac{m}{p} + 1 \leq r \leq \alpha k\frac{m}{p}$ **and**
 $1 \leq s \leq \frac{n}{p}$
 compute $S(r, s)$;
 (1.2.3) **if** $i \neq p$ **then**
 send (R_i^k, P_{i+1}) ;
-

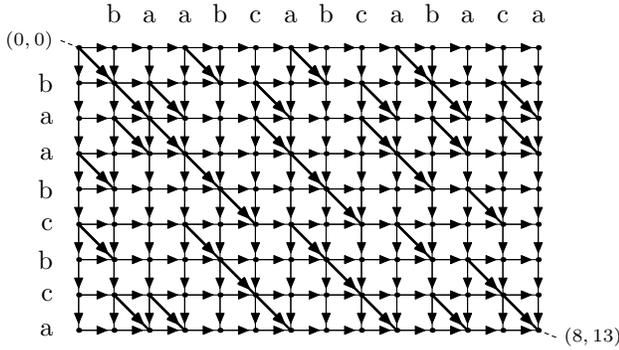


Figure 6. GDAG for the LCS problem, with $X = \text{baabc bca}$ and $Y = \text{baabcabcabaca}$.

Given the string *lewis Carroll*, an example of a subsequence is *scroll*. Given two strings X and Y , the *longest common subsequence* (LCS) problem finds the length of the longest subsequence that is common to both strings. If $X = \text{twasbrillig}$ and $Y = \text{lewis Carroll}$, the length of the longest common subsequence is 5 (e.g. *warll*).

Sequential algorithms for the LCS problem are surveyed in [7, 14]. It can be solved sequentially in $O(mn)$ time. PRAM algorithms for LCS are presented in [11]. The LCS problem can be solved on a PRAM [11] in $O(\log n)$ time with $mn/\log n$ processors, when $\log^2 m \log \log m \leq \log n$. In the following we show that a parallel CGM algorithm similar to the string alignment algorithm of the previ-

ous section can be easily derived.

As in the string alignment problem, the longest common subsequence (LCS) problem can be modeled by a *grid directed acyclic graph* (GDAG). Consider two strings X and Y of lengths m and n , respectively. Let $X = \text{baabc bca}$ and $Y = \text{baabcabcabaca}$. The corresponding GDAG has $(m+1) \times (n+1)$ vertices (see Fig. 6). We number the rows and columns starting from 0. All the vertical and horizontal edges have weight 0. The edge from vertex $(i-1, j-1)$ to vertex (i, j) has weight 1 if $x_i = y_j$. If $x_i \neq y_j$, this edge has weight 0 and can be ignored.

Thus the same ideas of the previous section apply here and we have a parallel algorithm for LCS that is similar to DP-score or DP-align of the previous section.

6. Experimental Results

We have implemented the proposed comparison strategy to compare the two genomes *Xanthomonas axonopodis* pv. *citri* (*Xac*) and *Xanthomonas campestris* pv. *campestris* (*Xcc*). The parallel platform used is a Beowulf cluster of 64 nodes consisting of low cost microcomputers with 256 MB RAM, 256 MB swap memory, CPU Intel Pentium III 448.956 MHz, 512 KB cache, in addition to two access nodes consisting of two microcomputers each with 512 MB RAM, 512 MB swap memory, CPU Pentium 4 2.00 GHz, and 512 KB cache.

The cluster is divided into two blocks of 32 nodes each. The nodes of each block are connected through a 100 Mb fast-Ethernet switch. Each of the access nodes is connected to the switch that connects the block of nodes and the two switches are connected. Our code is written in standard ANSI C using the LAM-MPI library.

The input data is obtained from Genbank (National Center for Biotechnology Information of the National Institute of Health [12]). The following is a summary of the input:

- *Xanthomonas axonopodis* pv. *citri* (*Xac*):
5,175,554 base pairs and 4,313 protein-coding genes. The main files used were NC_003919.fna, NC_003919.ptt and NC_003919.faa.
- *Xanthomonas campestris* pv. *campestris* (*Xcc*):
5,076,187 base pairs and 4,182 protein-coding genes. The main files used were NC_003902.fna, NC_003902.ptt and NC_003902.faa.

We summarize the implementation results as follows. For both phases we used 64 processors.

6.1. Phase I - Find Pairs of Homologous Genes

This is a very time consuming phase. All the genes of one genome are compared to all the genes of the other genome, giving rise to a total of more than 18 million comparisons.

We have exploited parallelism in two ways. First a standard master-slave approach is used where a master processor distributes the many comparison tasks to the slave processors. Second we use parallelism to find the similarity between two sequences, whenever the sequences are longer than a certain size (in this case 5,000 base pairs). To compare such sequences we used the parallel algorithm of Section 4.

The parallel solution performed this phase in one hour and 15 minutes, including computation and communication. The solution of [13] took about 3 hours. This gain does not seem to be so significant. However, it must be emphasized that, as described in our comparison strategy, we used a dynamic programming approach to compare the sequences as opposed to the method used in [13] where Blast [2, 3] and EGG [1] were used. Blast and EGG use heuristics and are less accurate, in a biological point of view, than the dynamic programming approach. Therefore the results we obtain are better in quality. Thus it is not meaningful to compare the time by Blast/EGG and our time by dynamic programming. For example it is not adequate to say that we obtain only a gain of 2.4.

We did not run the experiment by using only one processor because of memory and time constraints. So we do not present a speedup of the parallel solution relative to the sequential one. Notice however that we gain by nearly a factor of 64 in that part where comparisons are done by the slaves, since that part is trivially parallelizable. Furthermore, we also gain when we use the parallel algorithm for comparison of sequences longer than 5,000.

6.2. Phase II - Find the Non-Crossing Pairs

We used the parallel LCS - Parallel Longest Common Subsequence algorithm (shown in the previous section). Actually we could have used a less expensive approach, namely the LIS - Longest Increasing Subsequence [8]. In a practical viewpoint this is relatively cheap phase. Our parallel LCS solution took only 20 seconds. Of course this can be improved if we used the less expensive LIS approach.

6.3. Phase III - Compare the Intergenic Regions

The third phase compared the intergenic regions corresponding to the gaps between two consecutive homologous genes in each genome. The parallel solution took less than 3 minutes to compute the similarity of the intergenic regions. This phase has not been performed in previous works and so comparison is not possible. The relevance of this phase in a biological viewpoint is yet to be investigated.

7. Conclusion

With recent advances of DNA sequencing technology, the genomes of many organisms have been sequenced. The amount of genome information already available is astounding. In this area so overwhelmed by data and incomplete information we are urgently in need of high-performance computing tools to analyze and interpret the vast amount of raw data and fragments of information. Comparison of the DNA sequences and genes of two whole genomes can be useful to investigate the common functionalities of the corresponding organisms.

In this paper we use high-performance parallel computing to compare the whole genomes of two organisms of the genus *Xanthomonas*, namely *Xanthomonas axonopodis* pv. *citri* and *Xanthomonas campestris* pv. *campestris*.

We have implemented the proposed comparison strategy to compare the two genomes each consisting of more than 5 million base pairs and four thousand protein-coding genes. The parallel platform used is a Beowulf cluster of 64 nodes consisting of low cost microcomputers.

The most time consuming phase is to find homologous genes, with a total of more than 18 million pairs of genes to be compared. We have exploited parallelism in two ways. First a standard master-slave approach is used to distribute the many comparison tasks to the slave processors. Second we use parallelism to find the similarity between two sequences, whenever the sequences are longer than a certain size (in this case 5,000 base pairs). The parallel solution performed this phase in one hour and 15 minutes. The solution of [13] took about 3 hours. While this gain does not seem to be so significant, it must be noted that we used a dynamic programming approach to compare the sequences as opposed to the method using Blast and EGG. Our results are more accurate and give richer biological information. The more costly dynamic programming approach was made possible by the use of high performance parallel computation.

It can be argued whether it is worthwhile to spend more time by using the dynamic programming approach to get better quality results. We argue that the longer time is worth the effort for the sake of richer biological information if this time is tolerable, which is clearly the case in this parallel approach.

It must also be emphasized that we do not wish merely an alignment that reflects brute similarity between two genome sequences, but rather an alignment that takes into account the functional information. This was obtained by considering the homologous genes found between the two genomes.

Our comparison strategy also compares the intergenic regions corresponding to the gaps between two consecutive homologous genes in each genome. The relevance of this in a biological viewpoint is yet to be investigated.

The purpose of the paper is to show that with a high performance parallel approach, it is possible to explore new horizons and tread on new terrain, eventually paving the way for new discoveries.

Acknowledgments

The authors wish to thank the anonymous referees for their helpful comments.

References

- [1] N. F. Almeida Jr. Tools for genome comparison. *Ph. Thesis - IC/Universidade Estadual de Campinas*, May 2003.
- [2] S. F. Altschul, W. Gish, W. Miller, E. W. Myers, and D. J. Lipman. A basic local alignment search tool. *Journal of Molecular Biology*, 215(403):403–410, 1990.
- [3] S. F. Altschul, T. L. Madden, A. A. Schaffer, J. Zhang, Z. Zhang, W. Miller, and D. J. Lipman. Gapped blast and psi-blast: a new generation of protein database search programs. *Nucleic Acid Research*, 25:3389–3402, 1997.
- [4] C. E. R. Alves, E. N. Cáceres, F. Dehne, and S. W. Song. A CGM/BSP parallel similarity algorithm. In *Proceedings I Brazilian Workshop on Bioinformatics*, pages 1–8, Oct. 2002.
- [5] C. E. R. Alves, E. N. Cáceres, F. Dehne, and S. W. Song. Parallel dynamic programming for solving the string editing problem on a CGM/BSP. In *Fourteenth ACM Symposium on Parallel Algorithms and Architectures - SPAA 2002*, pages 275–281. ACM, Aug. 2002.
- [6] C. E. R. Alves, E. N. Cáceres, F. Dehne, and S. W. Song. A parallel wavefront algorithm for efficient biological sequence comparison. In V. Kumar, M. L. Gavrilova, C. J. K. Tan, and P. L'Ecuyer, editors, *The 2003 International Conference on Computational Science and its Applications - ICCSA 2003*, volume 2667 of *Lecture Notes in Computer Science*, pages 249–258. Berlin, Springer-Verlag, May 2003.
- [7] A. Apostolico and C. Guerra. The longest common subsequence problem revisited. *Algorithmica*, 2:315–336, 1987.
- [8] C. Cérin, C. Dufourd, and J. F. Myoupo. An efficient parallel solution for the longest increasing subsequence problem. In *International Conference on Computing and Information*, pages 220–224, Dec. 1993.
- [9] DIMACS. Workshop on whole genome comparisons. DIMACS Center, Rutgers University, Feb. 2001.
- [10] D. Gusfield. *Algorithms on Strings, Trees, and Sequences. Computer Science and Computational Biology*. Cambridge University Press, 1997.
- [11] M. Lu and H. Lin. Parallel algorithms for the longest common subsequence problem. *IEEE Transactions on Parallel and Distributed Systems*, 5(8):835–848, 1994.
- [12] NCBI/NIH. National Center for Biotechnology Information: GenBank growth statistics. <http://www.ncbi.nih.gov/>, 2003.
- [13] A. C. Rasera da Silva, J. C. Setubal, N. F. Almeida, et al. Comparison of the genomes of two *Xanthomonas* pathogens with differing host specifications. *Nature*, 417(6887):459–463, May 2002.
- [14] C. Rick. New algorithms for the longest common subsequence problem. Technical Report 85123–CS, Institut für Informatik, Universität Bonn, 1994.
- [15] J. C. Setubal and N. F. Almeida. Detection of related genes in prokaryotes using syntenic regions. In *DIMACS Workshop on whole genome comparison*, Feb. 2001.
- [16] J. C. Setubal and J. Meidanis. *Introduction to Computational Molecular Biology*. PWS Publishing Co., 1997.
- [17] TIGR. The Institute for Genomic Research. <http://www.ncbi.tigr.org/>, 2003.
- [18] D. W. Wood, J. C. Setubal, N. F. Almeida, et al. The genome of *Agrobacterium tumefaciens*: insights into the evolution of a natural genetic engineer. *Science*, 294(2323):2317–2323, Dec. 2001.
- [19] D. W. Wood, J. C. Setubal, N. F. Almeida, et al. Sequencing and analysis of the *Agrobacterium tumefaciens* genome. In *10th International Congress on Molecular Plant-microbe Interactions*, Dec. 2001.