Whole Genome Alignment using a Multithreaded Parallel Implementation

Wellington S. Martins[1], Juan del Cuvillo[1] Wenwu Cui[2] and Guang R. Gao[1] *
[1] Department of Electrical and Computer Engineering, University of Delaware
Newark, DE 19716 USA
{martins,jcuvillo,ggao@capsl.udel.edu}
[2] Department of Biological Sciences, University of Delaware
Newark, DE 19716 USA
{wcui@udel.edu}

Abstract—
Alignment of long DNA sequences is a challenging task due to its high demands for computational power and memory. We have developed a multithreaded parallel implementation of a sequence alignment algorithm that is able to align whole genomes with reliable output and reasonable cost. The implementation is based on a fine-grain multithreaded execution model, the EARTH model, which effectively tolerates latency through the overlapping of computation and communication. Human and mice mitochondrial genomes, human and *Drosophila* mitochondrial genomes are aligned respectively to demonstrate that the implementation can be used to align both closely related as well as less similar genomes. Results from *Mycoplasma genitalium* and *Mycoplasma pneumoniae* genomes, which are much larger than the tested mitochondrial genomes, are also presented. From the output, the homologous regions can be easily detected. This tool should facilitate alignment of syntenic regions, strain to strain comparisons, identification of regulatory elements and evolutionary comparisons as well.

Keywords— **Multithreading, Parallelism, Alignment, Genome, DNA.**

I. Introduction

Over the past few years a number of genomes have been completely sequenced by various research groups. A basic operation that can be immediately applied to this massive sequence data is the alignment of whole genomes. This approach of comparative genomics has a great biological significance. By aligning the DNA sequences of entire genomes (i.e. including coding and non-coding regions), scientists will be able to identify important matched and mismatched regions. The "matches" may turn out to be functional homolog pairs, conserved regulatory regions or long repeats. Various sizes of "mismatches" can be easily detected in the whole genome alignment. SNP (Single Nucleotide Polymorphism) is one base pair mismatch in the middle of two matching regions. Large sized "mismatches" may be foreign fragments inserted into the genome by transposition, sequence reversal or lateral transfer from another organism. This information will help to detect important functional differences between pathogenic and non-pathogenic strains of the same species, figure out evolutionary distance between organisms [LOO00], find out the regulatory regions of genes that may be preserved during evolution, and discover point mutations, deletions, reversions, insertions and duplications that may lead to diseases or special phenotypes.

Whole genome alignment can not be accomplished unless computer programs for pair-wise sequence comparison deal efficiently with both, execution time and memory requirements for this large-scale comparison. Some tools, requiring large amount of memory, have shown to achieve good execution times, but only at expense of accuracy. To find an alignment, they start by looking for perfect matches that can be further extended and joined together using a dynamic programming algorithm [SMI81]. For example, Kun-Mao et al. used a BLAST-like hashing scheme to identify exact *k-mer* matches and extend them to maximal-length matches [CHA95]. Delcher et al. applied a data structure called suffix-tree to find out, in linear time, perfect matches of a given length, also called *MUMs* [DEL99]. Maximal-length matches and MUMs are finally combined into local alignment chains by a dynamic programming step. However, neither of them apply the dynamic programming algorithm along the entire sequences since it is computational intensive — the algorithm's complexity is $O(n^2)$. Thus these tools are only applicable to closely related genomes.

We developed a parallel implementation of the dynamic programming algorithm that, by using the collective memory of several nodes, meets the computer memory requirements of this kind of application and is able to align any related genomes with reliable output in a reasonable time. The implementation runs on top of parallel machines based on off-the-shelf microprocessors, such as Beowulf installations (a cluster of standard PCs running Linux), and takes advantage of fine-grain multithreading to efficiently overlap computation and communication [THE99], producing impressive absolute speedups on Beowulf systems.

*This work was supported partially by the Delaware Biotechnology Institute (DBI). The work on EARTH is partially supported by DARPA, NSA, and NASA through the HTMT project; NSF (grants CISE-9726388, MIPS-9707125, EIA-9972853, and CCR-9808522); and DARPA through the DIVA project.

The rest of the paper is organized as follows. In section II, we review a pairwise sequence comparison algorithm using the dynamic programming technique. Our multithreaded parallel implementation is described in section III. We present results for real genomic sequences in section IV, and our conclusions in section V.

II. Pairwise Sequence Comparison Using Dynamic Programming

To compare two sequences we need to quantify the similarity between the pairs of symbols, one from each sequence, and associate a score for each possible arrangement. The measure of similarity of the two sequences is then given by the highest score. For example, let X = ATAAGT and Y = ATGCAGT and assume a score of 1 to matches and -1 to mismatches. By writing one sequence above the other we have the following possible alignment of the sequences:

```
sequence X   A  T  A  A  G  T
sequence Y   A  T  G  C  A  G  T
SCORE        1  1 -1 -1 -1 -1 -1   TOTAL = -3
```

To take the positions of the symbols into account we can consider sliding one sequence along the other so as to allow more symbols to match. In our example, shifting sequence X one position to the right results in a better alignment, i.e. one which produces a better score.

```
sequence X      A  T  A  A  G  T
sequence Y    A  T  G  C  A  G  T
SCORE        -1 -1 -1 -1  1  1  1   TOTAL = -1
```

However, these simple methods do not provide good measures of similarity when the sequences represent biological data (proteins or nucleotides). This is because biological sequences are a result of an evolutionary process in which mutations, i.e. substitutions, insertions or deletions, are bound to occur. Such mutations can be modeled by the introduction of gaps in the sequences. Assuming gaps score -2, our example can be modified to:

```
sequence X   A  T  A  -  A  G  T
sequence Y   A  T  G  C  A  G  T
SCORE        1  1 -1 -2  1  1  1   TOTAL = 2
```

The introduction of a gap (-) indicates a possible evolution from sequence X to sequence Y by the insertion of C. Alternatively, sequence Y might have evolved into sequence X by the deletion of C. Note that the third aligned pair, from left to right, can be understood as a mutation of A into G or, alternatively, of G into A.

When gaps are considered, the problem of sequence comparison becomes complex. Waterman showed that, given two sequences of length n, there are approximately $(1 + \sqrt{2})^{2n+1} n^{-1/2}$ possible alignments between these two [WAT89]. Now, if we consider real world genomic sequences whose size range from hundred thousands to hundred billions of base pairs, it is hopeless to enumerate all possible alignments. Thus, other methods must be used to find a solution to the alignment problem in less time.

In 1970 Needleman and Wunsch [NEE70] introduced the first algorithm for finding global alignments without enumerating all possible solutions. Global alignment attempts to match all of one sequence against all of the other. This algorithm was later adapted, by Smith and Waterman [SMI81], to the problem of local alignment, which finds alignments of subsequences of the two sequences. These algorithms consist of two parts: the calculation of scores indicating the similarity between the two given sequences, and the identification of the alignment(s) that lead to such score(s).

In order to avoid enumerating all possible alignments, these algorithms use a technique called *dynamic programming* [HOR78]. The idea is to build up the solution by using previous solutions for smaller subsequences. Thus, instead of recalculating the same value several times, the algorithm stores values, corresponding to partial results, in a data structure and reuses them as the new values are calculated. The data structure used is a two dimensional array which is called *similarity matrix*. This matrix is used to represent all possible alignments that can be constructed from the two inputed sequences.

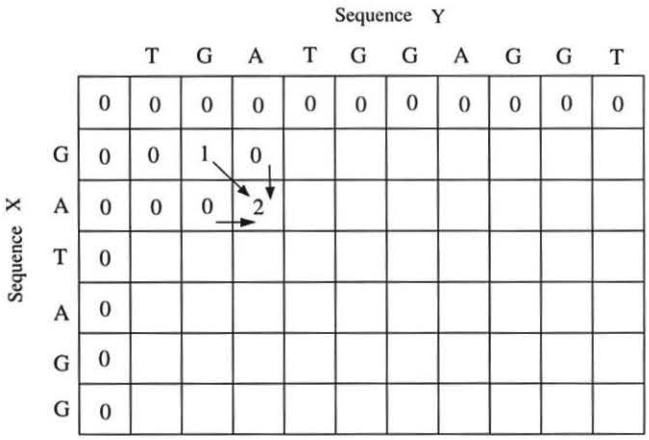

Fig. 1. A few steps of the calculation of the similarity matrix

The comparison of two sequences, X and Y, using the dynamic programming technique is illustrated in Figure 1. The sequences are placed along the left margin (X) and along the top (Y). The matrix is initialized with zeros along the first row and first column so that alignments between subsequences are not penalized by gaps on its left and right ends.

The other elements of the matrix are calculated by finding the maximum value among the following four values: left element plus gap penalization, upper-left element plus

the score of substituting the horizontal symbol for the vertical symbol, upper element plus the gap penalization, and zero (this condition forces the beginning of a new alignment if the score was to drop to a negative value). For example, the score 2 (3rd row and 4th column) is obtained by finding $max\{0+(-2), 1+(1), 0+(-2), 0\} = 2$. Notice that there are three possible alignments to be chosen from when calculating one element: alignment of the symbol in the row considered with a gap, alignment of the symbol in the row considered with the symbol in the column considered (either a match or a mismatch), and alignment of the symbol in the column considered with a gap. This corresponds to a horizontal move, diagonal move and vertical move between elements of the similarity matrix.

For the general case where X = $x_1, \ldots, x_i$ and Y = $y_1, \ldots, y_j$, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$, the similarity matrix $SM(n, m)$ is built by applying the following recurrence equation, where *gp* is the gap penalization and *ss* is the substitution score (match of mismatch).

$$SM[i,j] = max \begin{cases} SM[i, j-1] + gp \\ SM[i-1, j-1] + ss \\ SM[i-1, j] + gp \\ 0 \end{cases}$$

Following this recurrence equation, the matrix is filled from top left to bottom right with entry (i, j) requiring the entries $(i, j-1)$, $(i-1, j-1)$, and $(i-1, j)$. By choosing the maximum value we make sure the best score is found and stored, so that the next entries are build up based on that.

Once the similarity matrix is computed, the second part of the algorithm identifies the local alignments. Since the number of alignments grows exponentially, only alignments with score value above a given threshold are reported. Thus, for each such matrix element a trace-back procedure is applied to find out the actual base pairs that constitute the alignment. Starting at the end of the alignment and moving backwards to the beginning, this procedure follows a path like the ones described by arrows in Figure 2. Such path is determined at each cell considering its score and how it was produced. In other words, the maximum term of the recurrence equation determines which symbol (nucleotide) or gap are added to the alignment.

Although able to report all possible alignments between two sequences, sequence alignment algorithms based on the dynamic programming technique present a serious challenge. They impose important requirements both on computer memory and execution time. When dealing with long input sequences such as whole genomes, meeting these requirements is not a simple task. In the next section, we explain how our implementation meets the computer memory requirements and complete the task in reasonable time by means of parallelization.

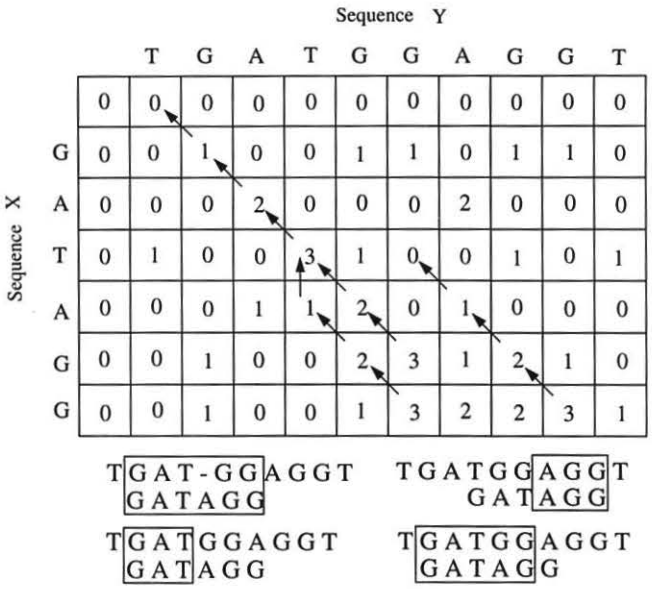

Fig. 2. Local alignments with score greater than 2

III. Parallel Computation of Sequence Alignment

A parallel version of the sequence comparison algorithm using dynamic programming must handle the data dependences presented by this method, yet it should perform as many operations as possible independently. Martins et al. [MAR01] showed that an efficient parallel implementation of the similarity matrix calculation can be done using multithreading. The implementation described in this paper builds on that and further applies parallelism to report the most significant local alignments of the input sequences. Before presenting our implementation we briefly describe, EARTH, the parallel execution model used.

A. The EARTH execution model

EARTH [HUM96, THE99] supports a multithreaded program execution model in which a program is viewed as a collection of threads whose execution ordering is determined by data and control dependences explicitly identified in the program. Threads, in turn, are further divided into *fibers* which are non-preemptive and scheduled according to dataflow-like firing rules, i.e., all needed data must be available before it becomes ready for execution. Programs structured using this two-level hierarchy can take advantage of both local synchronization and communication between fibers within the same thread, exploiting data locality. In addition, an effective overlapping of communication and computation is made possible by providing a pool of ready-to-run fibers from which the processor can fetch new work as soon as the current fiber ends and the necessary communication is initiated.

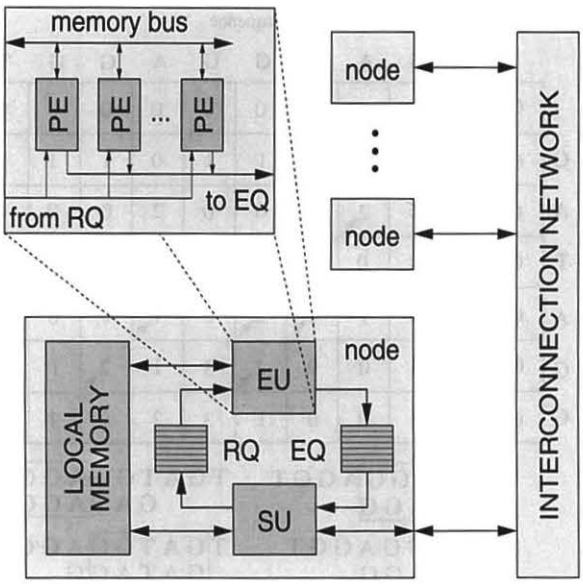

Fig. 3. EARTH architecture

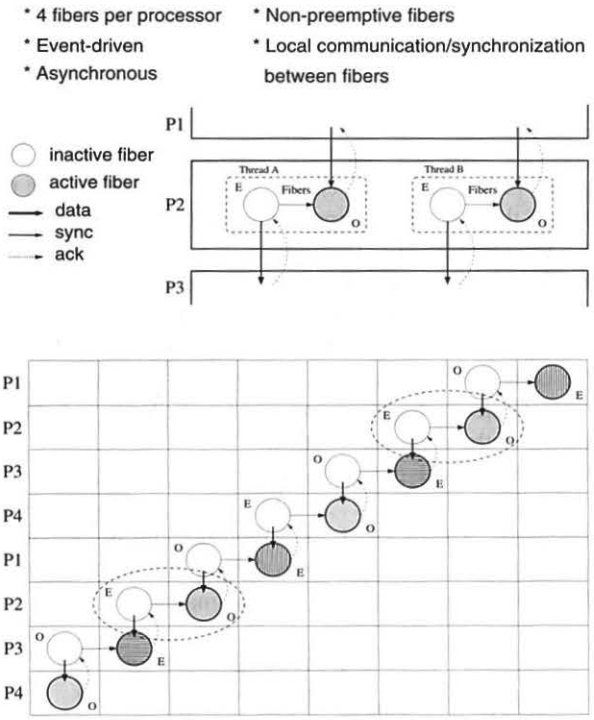

Fig. 4. Computation of the similarity matrix on EARTH

The EARTH model defines a common set of primitive operations required for the management, synchronization and data communication of threads. Each node in an EARTH system consists of an execution unit (EU), a synchronization unit (SU), queues linking the EU and SU, local memory, and an interface to interconnection network, see Figure 3. While the EU merely executes fibers, i.e., does the computation, the SU is responsible for scheduling and synchronizing threads, handling remote accesses and performing dynamic load balancing.

Although designed to deal with multiple threads per node, the EARTH model does not require any support for rapid context switching (since fibers are non-preemptive) and is well-suited to running on off-the-shelf processors. EARTH systems have been implemented on a number of platforms: MANNA and PowerMANNA, IBM SP2, Sun SMP cluster and Beowulf. EARTH programs are written using the programming language Threaded-C [HUM96, THE99]. This is an extension of the ANSI-C programming language which, by incorporating EARTH operations, allows the user to indicate parallelism explicitly.

B. A multithreaded parallel implementation

Our multithreaded implementation divides the scoring matrix into strips and each of these, in turn, into rectangular blocks. Generally speaking, it assigns the computation of each strip to a thread, having 2 independent threads per node. However, in order to better overlap computation and communication, blocks on a strip are actually calculated by two fibers within a thread. These fibers are repeatedly instantiated to compute one block at a time, and only one of the two fibers of each thread can be active at a particular time.

The decision of having two alternating fibers within each thread was based on the following reasoning. It would be a waste of resources if we had one separate fiber for each block, in each strip, since only one block can be calculated at a time. Having just one fiber for all blocks is also not a good idea because this fiber would get delayed due to the synchronization signal coming from the fiber immediately below. This signal acknowledges the receipt of data — without it the fiber, re-instantiated, would be allowed to overwrite the previous data. Thus, with just one fiber, computation would not be allowed to proceed until this acknowledgment signal is received. With the addition of an extra fiber we can further overlap computation and communication since one of the fibers can wait for the acknowledgment while the other starts working on the following block. (This double-buffering and acknowledgment scheme is used with other parallel applications on EARTH [THE00, THE99].)

A snapshot of the computation of the similarity matrix using our multithreaded implementation is illustrated in Figure 4. A thread is assigned to each horizontal strip and the actual computation is done by fibers labeled E(ven) and O(dd). The figure shows the computation of the main anti-diagonal of the matrix. The arrows indicate data and synchronization signals. For example, processor 2 sends data (downward ar-

rows) to processor 3 and receives data from processor 1 — i.e., fibers E of strips 2 and 6 send data to fibers E of strips 3 and 7, and fibers O of strips 1 and 5 send data to fibers O of strips 2 and 6. Fibers within a same thread, that is, associated with the same strip, send only a synchronization signal (horizontal arrows) since they share data local to the thread to which they belong. Finally, dotted upward arrows acknowledge the receipt of data so that the fiber receiving this signal can be re-instantiated to calculate another block of the same strip.

During the initialization phase, each thread grabs a piece of the input sequence X. This piece is all a thread needs from that sequence so the whole sequence need not be stored. Moreover, after computing a block, each fiber sends to the fiber beneath a piece of the sequence Y being compared. By doing so, we minimize the initialization delay that occurs when the nodes are reading the sequences from the server. Besides that piece of the sequence, a fiber also sends to the fiber beneath the scores and other information for each cell on the last row, data arrows in Figure 4. In this way, alignments that cross processors' boundaries can be detected.

As seen in section II, the number of possible alignments grows exponentially with the length of the sequences compared. Therefore, we cannot simple report all the alignments, instead we are interested in selecting only alignments with high scores. On each node, as each strip of the scoring matrix is calculated, when a score above the given threshold is found, it is compared with the previous highest scores stored in an table. The number of entries in the table corresponds to the maximum number of alignments that a node can report. Among other information, the table stores the cells' position where an alignment starts and ends. This feature allows us to produce a plot of the alignments found. A point worth noticing is that high score alignments are selected as the similarity matrix is calculated, row by row, thus the whole matrix needs not be stored.

C. The computational platform used

The experiments described in this paper were carried out using the Beowulf implementation of EARTH and a Beowulf machine consisting of 64 nodes, each containing two 200MHz Pentium Pro processors — total of 128 processors — and 128MB of memory. The interconnection network for the nodes is switched 100Mbps ethernet.

Initially, experiments using sequences ranging from 30K to 900K nucleotides long were carried out to test the scalability of the implementation part that calculates the similarity matrix. The absolute speedups are reported in Figure 5. The lack of speedup for the 30kx30k run is simply because there is not enough work to keep all 64 nodes (128 processors) busy. However as the sequence sizes increase, the speedup approaches the optimal linear speedup. A sequential computation of the 900kx900k run takes days to complete but the same result can be obtained in a few hours in the Beowulf cluster. Memory usage is also a limiting factor for a sequential computation. In contrast, a parallel implementation evenly distributes the data across the nodes.

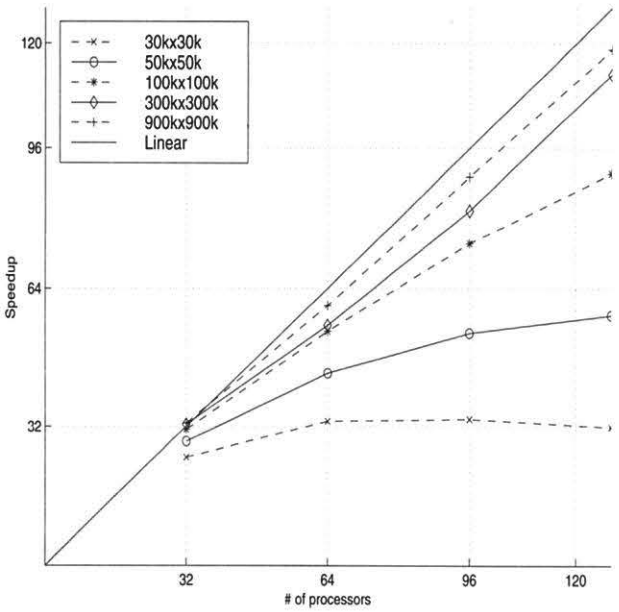

Fig. 5. Absolute speedup

TABLE I
EXECUTION TIME FOR THE M. PNEUMONIAE (816,394 NUCLEOTIDES) AND M. GENITALIUM (580,074 NUCLEOTIDES) GENOME COMPARISON

Implementation	Time
Seq. Smith-Waterman	53 hours
ATGC on 16 nodes	3.3 hours
ATGC on 32 nodes	2.1 hours
ATGC on 64 nodes	1.3 hours

In order to measure the performance and accuracy of our multithreaded implementation, further tests using genome sequences were conducted. Table I shows the execution times for both a sequential implementation of the Smith-Waterman algorithm and our parallel implementation running on the mentioned Beowulf system for the *Mycoplasma* genome comparison described in the following section.

IV. RESULTS

Our multithreaded parallel implementation, named ATGC — Another Tool for Genome Comparison, successfully aligned human and mice mitochondrial genomes and human and *Drosophila* mitochondrial genomes. It was also able to align *Mycoplasma genitalium* and

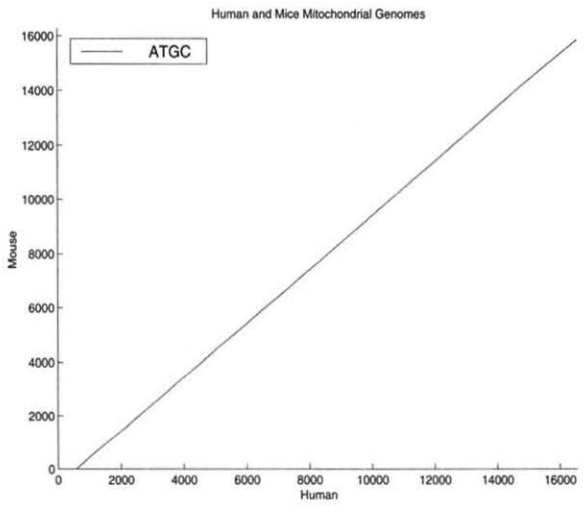

Fig. 6. Alignment of human and mice mitochondrial genomes generated by ATGC

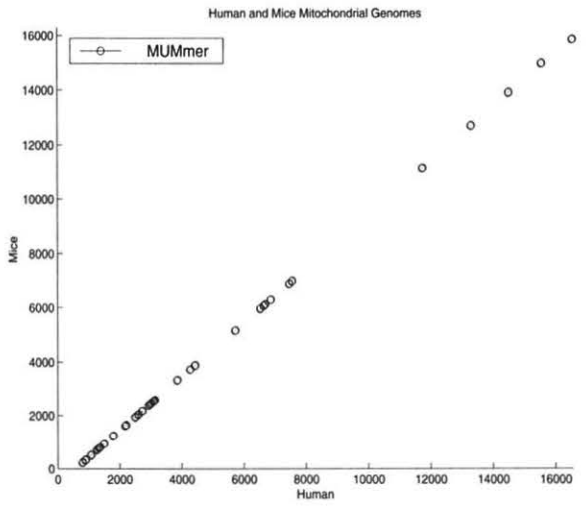

Fig. 7. Alignment of human and mice mitochondrial genomes generated by MUMmer

Mycoplasma pneumoniae genomes, which are much larger than the tested mitochondrial genomes, in a reasonable amount of time. All alignments were confirmed by MUMmer — a whole genome alignment tool (www.tigr.org/tigr-scripts/CMR2/webmum/mumplot), and *Blast2* — a tool for pairwise sequence comparison (www.ncbi.nlm.nih.gov/blast/bl2seq/bl2.html).

We used human, mice and *Drosophila* mitochondrial genomes to test the reliability of our Smith-Waterman parallel implementation, because first and second are somewhat related, and second and third are far related organisms. The alignments of these relatively small genomes can be easily verified by other tools. Further experiments were conducted by comparing larger genomes such as *Mycoplasma genitalium* (580,074 nucleotides) and *Mycoplasma pneumoniae* (816,394 nucleotides) genomes.

Alignment of human and mice mitochondrial genomes

The alignment of human and mice mitochondrial genomes generated by ATGC is showed in Figure 6. The straight line [1] confirms the similarity between these two genomes. This is also confirmed by MUMmer's alignment showed in Figure 7. However, MUMmer's alignment contains a gap between nucleotides 8,000 and 12,000 of human and mice mitochondrial genomes, whereas ATGC generates a single aligned segment. To investigate this discrepancy, we used Blast2 to align the aforesaid region. The alignment generated by Blast2, see Figure 8, showed that an important homolog pair, human and mice ATP synthase F_0 subunit was ignored by MUMmer but detected by ATGC. MUMmer missed that because there is no exact match equal or longer than 20 (MUM's — Maximal Unique Match, default size) in this region.

[1] A dot plot basically shows the similarity matrix computed during the first stage of the algorithm. The names of the sequences are placed along the X and Y axis. Each line/dot in the plot represents an actual alignment reported by the second phase of the algorithm

Alignment of human and Drosophila mitochondrial genomes

The mitochondrial genomes of Human and *Drosophila* are not as closely related as those of human and mouse. However, ATGC revealed some interesting similarity between these genomes (Fig. 9). Unfortunately, MUMmer could not confirm the alignment since only one exact match (MUM's size ≥ 20) is present, see Figure 10.

To confirm the alignment generated by ATGC is meaningful and reliable, Blast2 was used to search for similarity in human and *Drosophila* mitochondrial genomes. Between nucleotides 14,000 and 16,000 of human mitochondrial genome and 11,000 to 13,000 of *Drosophila* mitochondrial genome, which were aligned by ATGC, Blast2 found the Cytochrome B homolog pair in human and *Drosophila* (Fig. 11). In the region between base pairs 11,000 to 14,000 of human mitochondrial genome and 7,000 to 10,000 of *Drosophila* mitochondrial genome, where a gap was reported by ATGC, Blast2 did not find any significant similarity either (data not shown).

Alignment of Mycoplasma pneumoniae and Mycoplasma genitalium genomes

ATGC was applied to compare *Mycoplasma pneumoniae* and *Mycoplasma genitalium* genomes which are far more larger than mitochondrial genomes. This task is over 1,200 times more computational intensive than comparing the mitochondrial genomes we used above.

```
Query: human mitochondrial genome
Subject: mouse mitochondrial genome

8716 aaaggacgaacctgatctcttatactagta 8745
64    K  G  R  T  W  S  L  M  L  V
      |||||||||||| ||| | || ||| | ||
8116 aaaggacgaacatgaaccctaataattgtt 8145
64    K  G  R  T  W  T  L  M  I  V

8746 tccttaatcatttttattgccacaactaac 8775
74    S  L  I  I  F  I  A  T  T  N
      ||| ||||||| ||||||| |||| ||
8146 tccctaatcatatttattggatcaacaaat 8175
74    S  L  I  M  F  I  G  S  T  N

8776 ctcctcggactcctgcctcactcatttaca 8805
84    L  L  G  L  L  P  H  S  F  T
      ||||| || ||  | || ||  ||||||||
8176 ctcctaggccttttaccacatacatttaca 8205
84    L  L  G  L  L  P  H  T  F  T

8806 ccaaccacccaactatctataaacctagcc 8835
94    P  T  T  Q  L  S  M  N  L  A
      || || |||||||||||| ||||| |||
8206 cctactacccaactatccataaatctaagt 8235
94    P  T  T  Q  L  S  M  N  L  S
```

Fig. 8. Part of the alignment of human and mice ATP synthase F_0 subunit found by Blast2

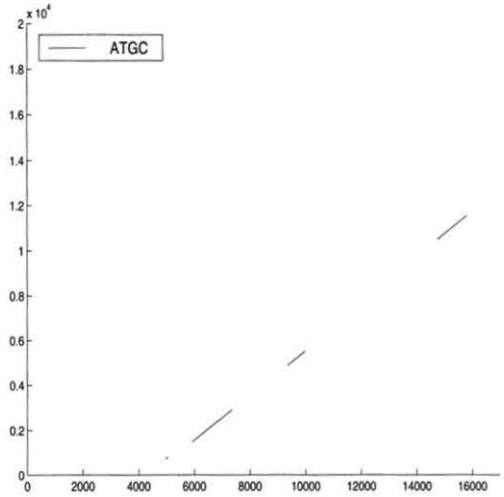

Fig. 9. Alignment of human and Drosophila mitochondrial genomes generated by ATGC

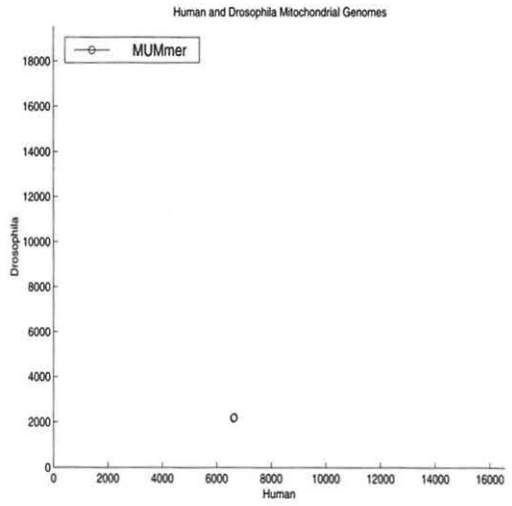

Fig. 10. Alignment of human and Drosophila mitochondrial genomes generated by MUMmer

```
Query: human mitochondrial genome
Subject: drosophila mitochondrial genome

15116 atagcaacagccttcataggctatgtcctc 15145
124    M  A  T  A  F  M  G  Y  V  L
       |||| |||||| || ||||| || ||  |
10870 ataggaacagcttttataggatacgtatta 10899
125    M  G  T  A  F  M  G  Y  V  L

15146 ccgtgaggccaaatatcattctgaggggcc 15175
134    P  W  G  Q  M  S  F  W  G  A
       || ||||| |||||||||||| ||||  ||
10900 ccttgaggacaaatatcattttgagtagct 10929
135    P  W  G  Q  M  S  F  W  V  A

15176 acagtaattacaaacttactatccgccatc 15205
144    T  V  I  T  N  L  L  S  A  I
       || || ||||| || ||| ||| ||| |||
10930 actgttattactaatttattatacgctatc 10959
145    T  V  I  T  N  L  L  Y  A  I

15206 ccatacattgggacagacctagttcaatga 15235
154    P  Y  I  G  T  D  L  V  Q  W
       || ||| | || | |||  ||||||||||
10960 ccttacttaggtatagatttagttcaatga 10989
155    P  Y  L  G  M  D  L  V  Q  W
```

Fig. 11. Part of the alignment of human and Drosophila Cytochrome B found by Blast2

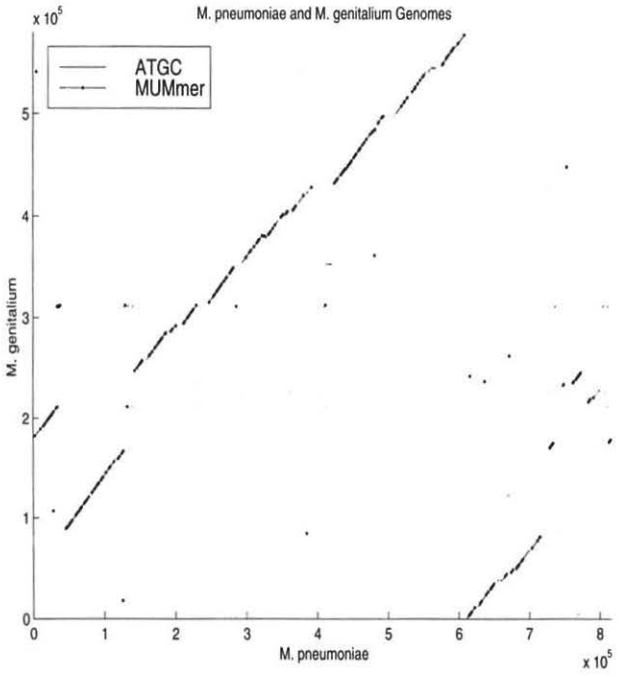

Fig. 12. Alignments of M. genitalium and M. pneumoniae genomes generated by MUMmer and ATGC

M. pneumoniae and *M. genitalium*, which belong to the same genus, are very closed related. Both ATGC and MUMmer generated similar results, see Figure 12. They aligned the two genomes successfully and clearly showed five translocations of *M. pneumoniae* with respect to *M. genitalium*, which is consistent with the results of Himmelreich et al. [HIM97].

V. Conclusions

Comparison of whole genome sequences can be done using traditional pair-wise sequence comparison algorithms based on dynamic programming, but doing so requires high computational and memory demands. We have developed a multithreaded parallel implementation of such algorithm that runs on cluster of PCs (Beowulf systems) and meet these requirements. The implementation produces accurate results in a reasonable amount of time, and uses the collective memory of the cluster to evenly distribute data, obviating the need for a machine with non-standard amount of memory.

The experimental results showed that our implementation aligns closely related and less similar genomes as well. Other genome comparison tools, based on different kinds of heuristics, complete the task fairly quickly, but at the expense of accuracy, as we have seen for the human and *Drosophila* mitochondrial genome comparison. So we believe our implementation can be an important complementary tool when higher accuracy is required on whole genome comparisons.

VI. Acknowledgements

We thank Michigan Technological University and Prof. Phil Merkey for providing us access to Ecgtheow, the Beowulf cluster used in this study. This paper benefited from many discussions with Sun Kim, from DuPont Central Research and Development (Experimental Station, Wilmington). We also would like to thank Chris Morrone for helping us with the EARTH Beowulf runtime system.

References

[CHA95] Kun-Mao Chao, Jinghui Zhang, James Ostell, and Webb Miller. A local alignment tool for very long DNA sequences. *Computer Applications in the Biosciences*, 11(2):147–153, 1995.

[DEL99] Arthur L. Delcher, Simon Kasif, Robert D. Fleischmann, Jeremy Peterson, Owen White, and Steven L. Salzberg. Alignment of whole genomes. *Nucleic Acids Research*, 27(11):2369–2376, 1999.

[HIM97] Ralf Himmelreich, Helga Plagens, Helmut Hilbert, Berta Reiner, and Richard Herrmann. Comparative analysis of the genomes of the bacteria *mycoplasma pneumoniae* and *mycoplasma genitalium*. *Nucleic Acids Research*, 25(4):701–712, 1997.

[HOR78] E. Horowitz and S. Sahni. *Fundamentals of Computer Algorithms*. Computer Science Press, 1978.

[HUM96] Herbert H. J. Hum, Olivier Maquelin, Kevin B. Theobald, Xinmin Tian, Guang R. Gao, and Laurie J. Hendren. A study of the EARTH-MANNA multithreaded system. *International Journal of Parallel Programming*, 24(4):319–347, August 1996.

[LOO00] G. G. Loots, R. M. Locksley, C. M. Blankespoor, Z. E. Wang, W. Miller, E. M. Rubin, and K. A. Frazer. Identification of a coordinate regulator of *interleukins* 4, 13, and 5 by cross-species sequence comparisons. *Science*, 288(5463):136–140, April 2000.

[MAR01] Wellington S. Martins, Juan B. del Cuvillo, Francisco J. Useche, Kevin B. Theobald, and Guang R. Gao. A multithreaded parallel implementation of a dynamic programming algorithm for sequence comparison. In *Proceedings of the Pacific Symposium on Biocomputing*, pages 311–322, Mauna Lani, Hawaii, January 3–7, 2001. World Scientific.

[NEE70] Saul B. Needleman and Christian D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48:443–453, 1970.

[SMI81] Temple F. Smith and Michael S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147:195–197, 1981.

[THE00] Kevin B. Theobald, Rishi Kumar, Gagan Agrawal, Gerd Heber, Ruppa K. Thulasiram, and Guang R. Gao. Developing a communication intensive application on the EARTH multithreaded architecture. In *Proceedings of the 6th International Euro-Par Conference*, number 1900 in Lecture Notes in Computer Science, pages 625–637, Munich, Germany, August–September 2000. Springer-Verlag.

[THE99] Kevin Bryan Theobald. *EARTH: An Efficient Architecture for Running Threads*. PhD thesis, McGill University, Montréal, Québec, May 1999.

[WAT89] Michael S. Waterman. *Mathematical Methods for DNA Sequences*. CRC Press Inc, Boca Ratón, Florida, 1989.