

A data-flow modification of the MUSCLE algorithm for multiprocessors and a web interface for it ¹

Alexey N. Salnikov (salnikov@cs.msu.su)

Faculty of Computational Mathematics and Cybernetics, Lomonosov Moscow State University, 2-nd educational building, Leninskie Gory, Moscow, 119992, Russia

Abstract Nucleotide and amino acid sequences research is actual for molecular biology and bioengineering. An important aspect of analysis of such sequences is multiple alignment. This article describes the implementations of the MUSCLE and ClustalW programs on multiprocessors and a web interface to them. The modification of the MUSCLE algorithm realize a data-flow manner of sequence alignment. It uses the PARUS system to build a data-flow graph and execute it on one multiprocessor. The data-flow algorithm has been tested on the sequences of human Long Terminal Repeats class five (LTR5) and several other examples.

Keywords. sequence alignment, data-flow, clusters

Introduction

Nucleotide and amino acid sequences research is actual for molecular biology and bioengineering. An important aspect of analysis of such sequences is multiple alignment. Its purpose is to arrange the sequences of DNA, RNA, or protein to identify regions of similarity that may be a consequence of functional, structural, or evolutionary relationships between the sequences. Aligned sequences are typically represented as rows within a matrix. Gaps are inserted between the residues so that residues with identical or similar characters are aligned in successive columns. There are several algorithms of multiple sequence alignment: ClustalW [1], MUSCLE [3], DIALIGN-TX [8]. The most popular method for multiple sequence alignment of nucleotides and amino acids is ClustalW. But ClustalW has several disadvantages and a new algorithm MUSCLE that tries to solve these disadvantages was developed. But on a large number of sequences or on long sequences the MUSCLE algorithm, originally non-parallel, can consume rather long time for the execution — up to several days, and can demand a large amount of memory. It should be noted that several algorithms has parallel implementation (for example: ClustalW-MPI [2], DIALIGN P [9]).

For the pairwise alignment the dynamic programming method [4] allows to get the precise result in a reasonable time, but for the multiple alignment the time complexity of the multidimensional analog of dynamic programming is an exponential function with

¹This work was supported by grants RFBR-08-07-00445-a and MK-1606.2008.9.

respect to the number of sequences. Usually heuristic algorithms are used, which do not give precise solution in the terms of optimization of some score function, but have a less complexity and give an eligible result in the context of biology. We will consider the algorithm described in the article [3]. It is realized in the MUSCLE software product available at <http://www.drive5.com/muscle>.

A web interface has been developed to facilitate availability of multiprocessor for researcher. The web interface requires creating the task by researcher. Each task binds the source data represented as sequences in FASTA format, algorithm (ClustalW-MPI, modified MUSCLE) and multiprocessor resources.

1. The MUSCLE algorithm

The MUSCLE algorithm is divided into several stages.

On the first stage the algorithm calculates the degrees of similarity between all aligned sequences and aggregates them into a matrix. Then the sequences are clustered by their similarities, and a binary cluster tree is built from the similarity matrix using the UPGMA[5] algorithm.

The similarity degree of sequences is calculated by the following method: all sequences are divided into sets of fragments with the length of k (k -mers), then the occurrence of each k -mer in all the sequences is calculated. Similarity calculation uses this formula:

$$similarity(S_i, S_j) = \sum_{m=1}^M \frac{\min(frequency(S_i, \tau_m), frequency(S_j, \tau_m))}{\min(length(S_i), length(S_j)) - k + 1}$$

where τ_m is one of the k -mers distinguished from the sequences, $frequency(S_i, \tau_m)$ is the number of times τ_m was met in sequence S_i , M — the number of all different revealed k -mers for all sequences in alignment.

After similarity calculating, the alignments for pairs of leaves having a common parent in the tree are built. The alignments for the pairs of sequences are built using a modification of the dynamical programming method. Then we get sub-alignments (alignments of sequences subset) for the all interior tree nodes that is done by pairwise alignment of two sub-alignments. Sequences that are involved in such sub-alignment cannot be shifted, so gaps are inserted simultaneously into all lines of sub-alignment. Quality of such sub-alignment are calculated by the following formula:

$$score(R_x, R_y) = \sum_{i=1}^L \sum_{\alpha} \sum_{\beta} f(\alpha, R_{x,i}) f(\beta, R_{y,i}) substitute(\alpha, \beta) - \sum_{j=1}^G gap_penalty(g_j), \quad (1)$$

$$\alpha \in column_symbols(R_{x,i}),$$

$$\beta \in column_symbols(R_{y,i}).$$

where $column_symbols(R_{x,i})$ determines the set of all the different symbols of a column i of sub-alignment R_x , and $f(\alpha, R_{x,i})$ calculates the frequency of α symbol occurrences in the column i of sub-alignment R_x . The function $gap_penalty(g_j)$ calculates the penalty for one individual gap where G is a number of all gaps which are inserted into the entire column, for either all rows of R_x or R_y .

On the second stage the tree is refined using the Kimura distance [3], a more accurate metric than the k-mer distance used on the first stage. Then the alignment is rebuilt by the method similar the first stage. Practically, algorithm needs to realign the sub-alignments that corresponds to the vertices which have been changed in tree.

The third stage is iterative. The alignment is “refined”.

2. The data-flow algorithm for multiple sequence alignment

For using the MUSCLE algorithm on multiprocessor systems, the original method has been modified on the stage of building sub-alignments using the cluster tree. We build a parallel program as a graph-program by means of PARUS [7](PARUS is a parallel programming language that allows to build parallel programs in a data-flow graph notation (<http://parus.sf.net>)).

PARUS allows user to create a file with text which describes vertices, edges, user defined C++ declarations and C++ code of prologue and epilogue. Then this file will be converted to the MPI/C++ source code using the PARUS utility called `graph2c++`. Once the source code is compiled to a binary executable file, this action is performed by using the standard MPI utility called `mpicxx`. This code is ready for running on any multiprocessor system or cluster (including multicore processor cluster). Edges and vertices implement the abstraction of data-flow. Hence, there are source vertices which do not contain any incoming edges and drain vertices which do not contain any outgoing edges. Other vertices could be connected by edges with each other. All vertices are converted to the C++ functions. Each function contains a code of vertex and service MPI-code generating automatically. The first action performed by all the MPI-process when the binary file is executing, is the call to the prologue routine. The prologue routine corresponds to the prologue code that is described in the graph-program source file. Then the code for all internal vertices will be executed in the order defined by edges, from the source vertices to the drain vertices. Vertices send MPI-messages each to other if they are connected by edge and appointed on executing to the different MPI-processes. Vertices are balanced between the MPI-processes to minimize execution time of vertex accordingly to the messages passing delays and processor performance. After all the drain vertices are finished the epilogue code will be executed by all the MPI-processes. Then the MPI-program is finished.

Developed data-flow algorithm is split into several stages. On the first stage we use the original MUSCLE algorithm to calculate similarities between sequences and to build the cluster tree. On the second stage we build a graph-program. The third stage concentrates on multiprocessor execution of the graph-program.

A graph-program is built from the cluster tree in the following way: each source vertex in a PARUS graph-program corresponds to one or several aligning sequences. Graph-program edges are directed from the source vertices to the inner ones. Inner vertices align pairs of sub-alignments to create a new sub-alignment that can be sent to a next inner

vertex or to a drain vertex of the graph-program. The algorithm goes to the drain vertex when the number of sequences in the current sub-alignment of global alignment is equal to the whole number of sequences to be aligned. The code of vertex induces the original muscle function to align a pair of sub-alignments.

Therefore, graph-program is built from the leaves of the cluster tree to the root vertex. To improve the efficiency of the parallel program the bottom (closer to leaves) layers of the cluster tree are compressed into one layer of graph-program. To do that, a new parameter is introduced into the algorithm. It defines a number of layers of the cluster tree to be compressed. As a result of the compression, the source vertices of the graph-program can contain more than one sequence, unlike the leaves in the cluster tree. For such source vertices the original MUSCLE algorithm is used to build a sub-alignments.

The time of program execution for the modified MUSCLE algorithm is reduced due to the possibility to execute each vertex of graph-program on a different processor. The maximal number of concurrently handled vertices is limited by the number of vertices in one layer of the graph-program and the number of processors accessible for running the parallel program.

3. Web interface

The web interface has been developed to bind needs of the researcher to align the sequences in a shortest time with a multiprocessor and job scheduling system on it (for example LoadLeveler).

Web interface is provided by simultaneous work of several programs: Python-script, bash-script, php-application. The php-application performs an interaction with user. The python script is automatically called by UNIX utility cron regularly (every hour for example). It looks for changes in user data or changes on one of the multiprocessors then it performs an actual actions. If one requires some interaction with multiprocessor, the python-script calls to bash-script on a remote machine via the SSH. The bash-script submits task to a queue on the multiprocessor, returns an information on task status, terminates task if necessary.

Let us to introduce an abstraction of user's task for the web interface. For each task, researcher should chose one of the multiprocessors, then for the chosen multiprocessor he specifies desired amount of time and number of processors or processor cores.

Once created task does not contain any sequences to align them. Such tasks have status *'new'*. User should upload sequences to the web-server via web interface, then is allowed to change task status on *'ready'*. After marking the task as *'ready'*, during one hour the sequences will be transmitted to the multiprocessor and the task will be submitted to the multiprocessor's queue. The task abstraction will be implemented on multiprocessor as the MPI-application executing on many processors.

Once per hour the task on the multiprocessor is automatically checked by means of Python-script. If the task finished its work, then the researcher will be notified by email on success or errors. There are two task statuses *'finished'* and *'refused'* in accordance with described situation on the multiprocessor. In any case, the result on success and the temporary data on error will be transferred back to the web site. Relations between task statuses are illustrated on the figure 1.

The web interface is available at: <http://angel.cs.msu.su/aligner>.

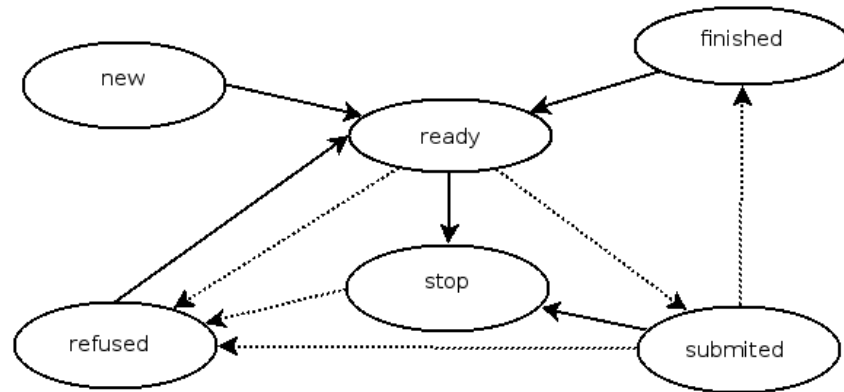


Figure 1. The graph of task statuses and rules to change them. The actions for rules denoted by solid arrows are performed by user. The actions for rules denoted by breaking arrows are performed automatically by the Python-script.

4. Testing results

The data-flow algorithm has been tested on sequences of LTR's class five in human genome. LTR (Long Terminal Repeat) is a family of so-called repeats (sequences occurring in the genome in a number of nearly identified copies); there are several classes of LTR. The class 5 (LTR5) contains approximately 1500 sequences approximately 1200 nucleotides each. The LTR5 collection described in [6]. Also multiple sequence alignment of 1088 protein sequences of globins approximately 300 amino acids each has been constructed.

We use several multiprocessor computers to build alignments. There are a one cluster and two SMP machines:

- PrimePOWER 850 is a SUN SMP machine with 12 processors
- IBM pSeries 690 "Regatta" is a SMP machine in 16 processors configuration (maximum is 32 processors).
- mvs100k is a cluster of 470 nodes with four Intel Xeon 5160 processors which are connected via Infiniband network.

Table 1. Delay values for LTR5

computer	single processor (min)	12 procs	16 procs
IBM pSeries 690	up to 420	21	24
PrimePOWER 850	68	28	-

Table 2. Delay values for protein sequences on mvs100k cluster

sequential code (sec)	16 cores	100 cores	500 cores
420	28	21	245

Table 3. Speedup values

number of procs (cores)	12	16	100	500
speedup	2,4	17,5	20	1,7

Results of tests are concentrate in tables: 1 and 2. The averaged over all machines and all types of input data values of algorithm speedup are concentrate in the table 3. We use a work time of the original one-processor MUSCLE algorithm with imposed limit of 3 iterations to estimate a speedup for parallel algorithm.

5. Discussion

Execution time of MPI-program that implements data-flow algorithm of multiple sequence alignment appreciably depends on several parameters. At first, it is very important for a cluster tree to be well balanced for maximizing parallel program efficiency. But cluster trees for biological sequences usually are rather imbalanced. Balanced tree provides maximum number of sub-alignments which could be aligned in parallel, because maximum number of vertices per level of cluster tree is achieved. Execution time depends on processors performance and volume of primary memory. For example Di-align and ClustalW-MPI has crash during they align LTR5 sequences on mvs100k cluster because 4Gb primary memory per cluster node is not enough for these algorithms.

Results show that it is important to search an optimal number of processors or processor cores if you align sequences using described data-flow algorithm. Unfortunately algorithm has not really good scalability. Further scalability improvement will be based on deep review of MUSCLE algorithm stages implementation. The maximal time complexity has an implementation of modified Needelman-Wunsch algorithm where quality of alignment is calculated by the formula (1). Parallel implementation of the Needelman-Wunsch algorithm is a very difficult task due to the type of dependencies current step of algorithm on the previous steps. So, probably, a dynamic programming step in MUSCLE algorithm for all stages will be parallelled. If there will be difficulties with Needelman-Wunsch then an other method for pairwise alignment building will be developed and parallelled.

References

- [1] Julie D. Thompson, Desmond G. Higgins, Toby J. Gibson CLUSTAL W: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position-specific gap penalties and weight matrix choice //Nucleic Acids Research, 1994, vol. 22 No. 22 , pp. 4673-4680. ISSN: 0305-1048 (Print), ISSN: 1362-4962 (Electronic).
- [2] Kuo-Bin Li ClustalW-MPI: ClustalW analysis using distributed and parallel computing //Bioinformatics Vol. 19, No. 12, 2003, pp. 1585-1586. ISSN: 1460-2059 (Electronic),ISSN: 1367-4803 (Print)
- [3] Robert C. Edgar MUSCLE: a multiple sequence alignment method with reduced time and space complexity //BMC Bioinformatics 2004, 5:113, ISSN: 1471-2105 (Electronic).
- [4] 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 Volume 48, Issue 3, 1970, pp. 443-453, ISSN: 0022-2836.
- [5] P.H.A. Sneath, Robert R. Sokal Numerical Taxonomy //Nature 193, pp. 855-860 (03 March 1962), ISSN: 0028-0836, EISSN: 1476-4687.
- [6] Alexeevski A.V., Lukina E.N., Salnikov A.N., Spirin S.A. Database of long terminal repeats in human genome: structure and synchronization with main genome archives //Proceedings of the fourth international conference on bioinformatics of genome regulation and structure, Volume 1. BGRS 2004, pp. 28-29 Novosibirsk.
- [7] Alexey N. Salnikov PARUS: A Parallel Programming Framework for Heterogeneous Multiprocessor Systems //Lecture Notes in Computer Science (LNCS 4192) Recent Advantages in Parallel Virtual Machine and Message Passing Interface, Volume 4192, pp. 408-409, 2006, ISBN-10: 3-540-39110-X ISBN-13: 978-3-540-39110-4.
- [8] Amarendran R Subramanian, Michael Kaufmann and Burkhard Morgenstern DIALIGN-TX: greedy and progressive approaches for segment-based multiple sequence alignment // Algorithms for Molecular Biology, 2008, 3:6, ISSN: 1748-7188 (electronic).
- [9] Martin Schmollinger, Kay Nieselt, Michael Kaufmann and Burkhard Morgenstern DIALIGN P: Fast pair-wise and multiple sequence alignment using parallel processors // BMC Bioinformatics, 2004, 5:128, ISSN: 1471-2105 (Electronic).