Parallel FSA: Improving the Performance of Multiple Sequence Alignment using a Workstation Cluster and Database

By
Jaeyoung Do

Submitted to the Department of Computer Sciences in fulfillment of the requirement for the Master project.

Advisor

Colin Dewey
Assistant Professor of Biostatistics and Medical Informatics & Computer Sciences

May 2009

UNIVERSITY OF WISCONSIN - MADISON
Parallel FSA: Improving the Performance of Multiple Sequence Alignment using a Workstation Cluster and Database

Jaeyoung, Do
University of Wisconsin-Madison
jae@cs.wisc.edu

Abstract
Multiple Sequence Alignments (MSA) are widely-used tools for biological sequence analysis such as function prediction and phylogeny inference. Recently, a MSA algorithm based on a statistically-sound method for model selection and parameterization, Fast Statistical Alignment (FSA), has been introduced. Although FSA is state-of-the-art with respect to accuracy and ability to scale to thousands of sequences, it still suffers from problems of long execution time and high memory consumption when trying to align truly large datasets of sequences. In this paper, we propose a parallelized version of FSA which interacts with a database to address these problems. First, it runs over a cluster of processors to reduce its runtime for pairwise comparisons. In order to restrict the memory to be consumed, it requests from the database only the required datasets while aligning sequences. We show that utilizing a cluster of processors can lead to large performance improvements. Also, by using a database, we can not only align large datasets without worrying about physical RAM limitations, but also, reuse the results of previous pairwise comparisons, resulting in a dramatic execution time reduction.

1. Introduction
Multiple sequence alignments (MSA) are important tools for protein, DNA and RNA sequence analysis. The field of MSA is very active, and as a result, there has been much effort spent on developing better algorithms. According to Wikipedia, there are approximately 60 available MSA programs, including ClustalW [1], which is now the most widely-used MSA program. More recently, a new MSA program named Fast Statistical Alignment (FSA) [2], based on a statistically-sound method for model selection and parameterization, has been suggested. Due to the number of available programs, it is not easy for biologists to select the best MSA program for their studies.

Generally, there are three main considerations [3] in choosing a MSA program: biological accuracy, execution time and memory consumption. Although biological accuracy is the most critical concern, as available sequence data increases rapidly, longer execution time and large memory consumption are now serious problems in practice. This is because the minimum required computational time and memory usage to maximize biological accuracy has been shown to scale exponentially with the number of sequences [4]. For this reason, most MSA programs have a practical limit on the number of sequences that can be aligned on modern desktop computers.

While FSA can align hundreds or thousands of sequences on a single computer, it also suffers from the problem of long execution time and high memory consumption when trying to align truly large sequence datasets. The goal of this paper is to address these large-scale problems of FSA with a cluster of processors and a database.

We first adopt parallelism to speed up FSA. This is possible because of a natural independence structure of FSA: each pairwise sequence alignment is independent of all other pairs, allowing a dramatic runtime reduction by distributing the individual pairwise computations to different processors. In order to make FSA parallelized, two factors are considered: communication overhead and workload distribution over different processors. For example, distributing jobs in very small batches may reduce processor idle time but lead to high overhead. In contrast, using large batches may increase idle time but minimize overhead. We experimented to find the optimal size of workloads to be distributed over the different processors, and found that the \textit{Fixed-Size Chunking} strategy [7] worked best: Each of the $P$ processors runs on chunks of $\frac{N \times (N - 1)}{2 \times P}$ pairwise comparisons, where $N$ is the number of sequences to be aligned.

In addition to taking a large amount of CPU time, FSA requires a sizable amount of memory for keeping pairwise probabilistic data computed via pairwise comparisons in RAM. Moreover, it should be useful to manage and store such data in a permanent storage medium for reuse. We handle these issues by storing some of the primary data structures in a database so that FSA can efficiently access the stored data during its execution. We believe that a database is good for managing such scientific data because of several of its features: schemas for metadata, parallelism for a distributed environment, and indices for speedup. These features are currently not provided by file-oriented approaches.
The rest of this paper is organized as follows. In section 2 we first present the related works in this field. We introduce the parallelization and the database mode of FSA in section 3 and 4, respectively. We show that the combination of the parallelization and the database mode of FSA ameliorates problems of long execution time and high memory consumption with experimental results in section 5. Finally, we conclude this paper and present future aims in section 6.

Note that throughout this paper, the term parallel FSA will be used to denote the parallel version of FSA and FSA for the original version (i.e., non-parallel version) of FSA.

2. Related Work

Many different MSA programs have been presented for aligning multiple biological sequences. These programs are mostly based on the idea of progressive alignment [18] that builds up a final alignment beginning with the most related pair, and progressing to the most distantly related pair. The first generation of MSA programs such as ClustalW [1] and T-Coffee [5], follow this approach. Progressive alignments, however, cannot be globally optimal because progressive methods are heuristics in which errors made at any stage in the course of aligning sequences are propagated through to the final result.

To address this inherent problem in progressive methods, another approach called iterative refinement, has been introduced [11]. This approach works similarly to progressive methods but repeatedly realigns selected subsets of sequences in order to achieve higher scores. Many well-known MSA programs, including DIALIGN [9] and MUSCLE [6], take this approach.

As another way to improve accuracy, a method based on a sequence annealing algorithm has been suggested [12]. This approach builds alignments one match at a time, matching positions that are more likely to be homologous.

Several state-of-the-art MSA techniques have offered significant improvements in biological accuracy, but not in scalability. For example, according to the latest benchmark on currently available MSA programs [3], T-Coffee has a practical limit of around 100 sequences on modern desktop computers because of speed and memory requirements.

For this reason, a variety of parallel versions of existing MSA programs have been presented for improving speed. For instance, parallel versions of ClustalW have been implemented on a workstation cluster using MPI [19, 20]. The DIALIGN program has also been parallelized [16]. Our work shares the same spirit with these works in striving to improve the speed of MSA programs. They, however, do not focus on another practical problem of MSA programs, high memory consumption required to align sequences, as we do.

3. Parallelization mode

3.1 Overview of the FSA Algorithm

For multiple alignment, the FSA algorithm works as follows: in the first step, all pairwise comparisons on a dataset of sequences are carried out to calculate the posterior probabilities. For \( N \) input sequences, by default, it performs an exhaustive distance-based alignment, which requires complete comparisons between all possible \( \frac{N \times (N-1)}{2} \) sequence pairs. Performing all pair-wise alignments takes \( O(N^2) \) time which gives unfavorable runtimes for datasets of many sequences. FSA overcomes this problem by reducing the number of pairs of sequences that are compared, costing \( O(N \log N) \). See [2] for the detailed explanation of this approach.

While many MSA programs use a model with parameters tuned for best performance on a particular dataset such as BAliBASE 3 [10], FSA estimates model parameters when calculating posterior probabilities by utilizing an unsupervised query-specific learning procedure. Although training parameters is a costly operation, it makes FSA robust. FSA uses different learning strategies for nucleotide and amino acid emission matrices. For RNAs and DNAs, FSA can learn a different emission distribution for every pairwise comparison of sequences. This is possible because of the small number of free parameters, \((4^2 – 1) = 15\). In contrast, FSA learns a single emission matrix using an all-pairs comparison for protein sequences since emission matrices over aligned amino acids have \((20^2 – 1) = 3,999\) free parameters.

After obtaining the pairwise posterior probabilities, FSA uses the sequence annealing technique [12] to construct a multiple alignment. This approach to alignment seeks to maximize the expected accuracy of the alignment using a steepest-ascent (greedy) algorithm.

3.2 Master-Worker Framework

For our parallel implementation, we use a modification of the approach in MW [15]. MW is a software framework to quickly and easily parallelize scientific computations using the master-worker paradigm on a cluster of processors. The master-worker approach works as follows. First, the master assigns tasks to workers, and then workers perform assigned tasks. Once tasks are completed, workers report results back to the master. This approach fits well with parallelizing FSA because the last stage of the FSA algorithm, the sequence annealing stage, runs on the master, and can only begin after collecting all pairwise posterior probabilities from workers.

MW also helps to manage the opportunistic environment of a cluster. Since, in general, processors in a cluster are not dedicated, one might suspend and resume its task at any time. MW handles such uncertainty issues automatically, as well as fault tolerance and task scheduling. Furthermore, it can manage the complexity of processor variations such as different architecture types, speed, OS, etc.

We modified MW for our purpose. We eliminated several unnecessary internal copies of network data, which were de-
grading performance, and the limit on the master’s buffer size, which restricted the largest size of a communication data chunk between the master and a worker to 16 MB. The small buffer size was not appropriate for our application because it is very common to send multi-GB probabilistic data from workers to the master. More importantly, when receiving probabilistic data from each worker, we have the master using POSIX threads so that all workers can concurrently transfer data to the master. The original version of MW does not support a multi-threaded environment, and therefore, while one worker is transferring data to the master, others who want to send data to the master must be blocked. We will explain the benefits of using multiple threads in section 3.3. We also added a recovery mechanism that guarantees re-distributing failed tasks under the multi-threaded environment.

Finally, note that instead of using MPI, the Message Passing Interface designed for a communication protocol of a cluster of processors, we directly used TCP/IP sockets to transfer data. Although MPI provides great communication functionalities between processors in a single cluster, it cannot be used beyond the cluster. Therefore, in order to put data into a database that might be outside of the cluster, we used TCP/IP sockets.

3.3 The workflow of parallel FSA

The primary goal of parallel FSA is to distribute tasks for computing all pairwise comparisons to available processors in a cluster. Because each pairwise alignment may be computed independently, it is quite possible to achieve near-linear speedup. To maximize this speedup, it is important to find the best strategy satisfying the following two concerns: First, the amount of message passing between the master and each worker should be minimized, and second, the total sum of idle times of workers should be minimized. These two concerns are related to load-balancing. That is, distributing tasks in very small batches may reduce processor idle time but lead to high overhead. In contrast, using large batches may increase idle time but minimize overhead. We experimented with several strategies, and got maximum speedup when distributing equal-sized batches, which is called the Fixed-Size Chunking strategy [7]. Therefore, each of the $P$ processors performs exactly $\frac{N \times (N-1)}{2 \times P}$ pairwise comparisons, where $N$ is the number of sequences to be aligned.

While the pairwise comparisons can be naturally parallelized, the sequence annealing stage does not have the same obvious independencies. Therefore, parallel FSA performs the sequence annealing stage only on the master. The parallelization of the annealing step is a future aim.

Figure 1 shows the schematic overview of parallel FSA. Once parallel FSA is run on the master, it trains parameters with all-pairs of sequences for protein if necessary (See section 3.1). Then it distributes the same number of sequence pairs to each of $P$ workers in a cluster with some extra data needed to calculate posterior probabilities. Each worker compares the sequence pairs transferred from the master to calculate posterior probabilities, training parameters for DNA and RNA sequences if necessary (See section 3.1).

Once a worker has completed the calculation, it first sends the finish signal to the master, and starts to send back calculated posterior probabilities. After collecting all pairwise posterior probabilities from workers, the master begins the sequence annealing.

When calculating pairwise posterior probabilities, threading is used on both the master and workers to maximize the performance of parallel FSA. For workers, threading is needed to send calculated posterior probabilities back to the master while simultaneously comparing sequence pairs and calculating pairwise posterior probabilities. If threading is not used, transferring calculated probabilities to the master can be begun only after finishing the comparisons of sequence pairs. If threading were not used, there would be nothing for workers to do while sending the data back, which usually takes a long time (Recall that the size of the...
data to be sent to the master from each worker is normally multi-GB). To address this problem, each worker maintains a 10MB size buffer to be filled with probability data. When the buffer is full, a thread is started to send the data in the buffer to the master, and make the buffer empty.

On the master, threading is used for establishing socket connections with all workers so that the master can get data back from the workers in parallel. In other words, if threading is not used, the master can only get the data back from workers sequentially.

Instead of sending computed posterior probabilities to the master directly via a TCP/IP socket communication, each worker might transfer them to a database for several reasons explained in the next section.

4. Database mode

FSA requires not only a long execution time because of its complex algorithm, but also high memory usage since it creates vast posterior probabilities as a result of pairwise comparisons for hundreds or thousands of sequences. This brings two concerns. First, it might be practically impossible to load all posterior probabilities in the RAM of a single computer. For example, to align 500 prokaryotic 16S sequences, 23.18 GB RAM is necessary, which is big enough to create out-of-memory problems on common workstations. Second, the total execution time can be dramatically reduced if appropriate pre-computed posterior probabilities are reused. Therefore, it is useful to store such huge amounts of probabilistic data in a secondary storage medium, and manage them efficiently for reuse.

4.1 Why use a Database?

A database is a better choice than pure file-oriented systems for storing and managing posterior probabilities because of three features: schemas for metadata, parallelism for a distributed environment and indices for fast lookups.

As mentioned in the previous section, we parallelized FSA by utilizing a cluster of processors to reduce its execution time. Therefore, no matter what approach we choose to store posterior probabilities, it must provide parallel processing functionalities so that processors can concurrently update data. Naturally, set-oriented file processing of a database enables parallelism by assigning a table to a processor. Furthermore, database researchers have actively studied ways to support CPU and I/O parallelism for over two decades, and this allows large corporations to manage 100 TB datasets today using a cluster of +1000 processors [17].

Accessing and validating stored posterior probabilities for reuse is enabled by Metadata. Metadata are structured data that describe characteristics of a resource. In our case, they explain the measured attributes, their names, units, accuracy, and ideally a great deal more. The latest version of FSA supports 20 parameter options that might change results of the pairwise posterior probability computations even with the same dataset of sequences. Therefore posterior probabilities must be stored with their metadata.

A database has a benefit in which it can express Metadata in a natural way. While file approaches only can convey Metadata through files and directory names, which are limited in their expression, a database uses schemas, which define tables, fields in each table, and relationships between the fields and the tables.

If the available memory of the working machine on which the sequence annealing is running is large enough to hold all posterior probabilities, parallel FSA transfers all of them from a database at its initial state for the best performance. Otherwise, it interacts with the database during the sequence annealing, requesting the necessary portions of the probabilistic data, sometimes in a sorted order on the fly. For this reason, it is required to find the relevant probabilistic data and sort them in an efficient way. Databases provide not
only powerful associative searches by value rather than by location of file-oriented approaches, but also carefully implemented external sorting algorithms that handle sizes of data that can not be fit in memory. These kinds of operations can be achieved, and significantly sped up by taking advantage of indexing.

4.2 Schema and Table

Figure 2 shows the tables and schemas maintained by parallel FSA to store and organize posterior probabilities. Note that we omitted some tables and fields of the presented tables for the purpose of clarity.

The main table stores information regarding different datasets of sequences. Each row of the table corresponds to a dataset, and contains fields for the number and the average length of sequences in the dataset and the hash value of the dataset. In order to compute a unique hash value of the dataset regardless of the order in which sequences are inputted, the sequences are first sorted based on their length and their ASCII character codes, and concatenated to make a long sequence string. Then, the hash value is computed with this string.

Each dataset of sequences has a schema to store sequence information and posterior probabilities, as well as the values of options used when generating the probabilistic data. The sequence table contains all information on the sequences in the dataset such as their strings, lengths and hash values. The parameter table maintains the values of parameter options and the number of processors utilized when generating posterior probabilities. As mentioned in section 4.1, even with the same dataset of sequences, results of pairwise comparisons might be varied based on what parameter option values are used. Therefore per each set of parameter options corresponding to a row in the parameter table, it is needed to manage different sets of probabilistic data, and the sets are stored in the probabilistic tables. If \( p \) processors are used to calculate the probabilistic data, the number of probabilistic tables assigned to a set of parameter options is also \( p \) since one table is assigned to each processor. This is done to improve the efficiency of parallelism by eliminating interference of concurrent updates to other processors.

To find and access the appropriate posterior probabilities, a combination of three fields in the main table, the number of sequences, average length of sequences, and the hash value of the set of sequences, is used to determine candidate datasets. In the case that more than one candidate dataset matches these criteria, all rows of the sequence table in the schema corresponding to each candidate dataset are examined.

4.3 Index

Parallel FSA utilizes indices to increase the speed of finding necessary portions of posterior probabilities in the database. During sequence annealing, the following two types of queries are most frequently requested.

- Find all posterior probabilities of a pair.
- Find \( N \) posterior probabilities in descending order.

These types of queries require efficient range searches and sorted results. In order to satisfy these two concerns we use two B+ tree indices on probabilistic tables, specifically on both \((seq \; i, \; seq \; j)\) fields and the prob field of each table.
Figure 4. Runtimes of parallel FSA when aligning 100 and 200 sequences of prokaryotic 16S, respectively. The paircomp time is the elapsed time of the pairwise comparison phase, and the annealing time is that of the annealing phase. The execution time is defined as the sum of the paircomp time and the annealing time.

5. Experiment

5.1 Test Environment

The parallelization and database results were achieved by experimental tests made on a cluster of processors managed by the Condor batch queuing system [13]. The cluster consisted of approximately 200 CPU cores, with the nodes interconnected via a 100 Mbps Ethernet network. For the database work, we used PostgreSQL [14] 8.3.1, a powerful open source relational database system. The database server was installed on a quad-core (each core running at 3 GHz) Intel Xeon machine with 16GB of RAM on which two 146 GB 15K RPM hard disks are installed.

Biologists commonly perform alignments of hundreds or thousands of 16S ribosomal DNA sequences in order to elucidate evolutionary relationships and build phylogenetic trees. We performed alignments of prokaryotic 16S sequences to demonstrate the effectiveness of the parallelization and the database modes. 16S sequences were obtained as a random slice of prokMSA from Greengenes [8] with an average length of 1,450nt.

5.2 Test Results

Figure 3 and 4 show the results of the parallelization mode test of parallel FSA. This experiment was run on a cluster with 20 processors (3.00 and 3.20 GHz; 8 GB of RAM).

Figure 3 shows speedups of the overall execution times and the pairwise comparisons when aligning 16S sequences. The test ran on 1, 5, 15 and 20 processors. The speedup of the sequence annealing phase is omitted since that phase is not parallelized.

Note that the speedup of the pairwise comparisons increases linearly with the number of processors. For example, using 10 processors, the speedup for both 100 seqs and 200 seqs was 9.6X. However, the curve of the overall speedup is not linear, but log-shaped. For instance, when 20 processors were utilized, the overall speedups for 100 seqs and 200seqs were only 12.69X and 6.15X, respectively.

As in Figure 4, the gap between the overall and the pairwise comparison speedups is because parallelizing the pairwise comparison phase has a little impact on the execution time of parallel FSA when using a large number of processors. In the case of using a small number of processors (e.g., 1 or 5 processors in Figure 4), the cost of the pairwise comparison phase is more expensive than that of the sequence annealing phase. As a result, the time reduction by parallelizing the pairwise comparisons has a significant impact on the overall speedup. However, as the number of utilized processors increases, the cost of the pairwise comparison phase is rapidly dropped below the sequence annealing cost. Consequently, the execution time converges to the sequence annealing time, which is independent of the number of processors.

Due to this reasoning, the maximum overall efficiency\(^1\) is likely to be achieved when a small number of processors is used for pairwise comparisons. In our experiment, the maximum overall efficiencies of 100 seqs and 200 seqs were about 90% and 68% with 5 processors.

As the next experiment, we tested the database mode of parallel FSA. This test was conducted on a 2 × quad-core

\[\text{Overall efficiency} = \frac{\text{Overall speedup}}{p}, \quad p \text{ is the number of processors}\]
Table 1. Memory consumptions and database disk usages of parallel FSA as the size of the dataset varied. The second column shows the required memory size to run parallel FSA on a single processor, and the third and fourth columns show the size of disk spaces needed to store all pairwise posterior probabilities when indices are used, and not used respectively.

<table>
<thead>
<tr>
<th>RAM</th>
<th>DB disk</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w/ index</td>
<td>w/o index</td>
</tr>
<tr>
<td>100 seqs</td>
<td>1,027 MB</td>
<td>2,565 MB</td>
</tr>
<tr>
<td>200 seqs</td>
<td>3,829 MB</td>
<td>10,441 MB</td>
</tr>
<tr>
<td>300 seqs</td>
<td>8,380 MB</td>
<td>23,387 MB</td>
</tr>
<tr>
<td>400 seqs</td>
<td>14,978 MB</td>
<td>42,044 MB</td>
</tr>
<tr>
<td>500 seqs</td>
<td>23,736 MB</td>
<td>67,566 MB</td>
</tr>
</tbody>
</table>

Table 2. Measured memory consumptions and runtimes of parallel FSA with restrictions on memory consumption. The second column presents the consumed memory size in the test. The third and fourth columns present the execution times when indices are, and not used respectively.

<table>
<thead>
<tr>
<th>RAM</th>
<th>Execution time</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>w/ index</td>
</tr>
<tr>
<td>100 seqs (-)</td>
<td>1,027 MB</td>
</tr>
<tr>
<td>100 seqs (700 MB)</td>
<td>684 MB</td>
</tr>
<tr>
<td>100 seqs (500 MB)</td>
<td>492 MB</td>
</tr>
<tr>
<td>200 seqs (-)</td>
<td>3,829 MB</td>
</tr>
<tr>
<td>200 seqs (3 GB)</td>
<td>2,986 MB</td>
</tr>
<tr>
<td>200 seqs (2 GB)</td>
<td>2037 MB</td>
</tr>
<tr>
<td>300 seqs (-)</td>
<td>8,380 MB</td>
</tr>
<tr>
<td>300 seqs (8 GB)</td>
<td>8,086 MB</td>
</tr>
<tr>
<td>300 seqs (4 GB)</td>
<td>4,023 MB</td>
</tr>
</tbody>
</table>

Table 1 contains memory consumptions for aligning different sizes of 16S datasets on a single processor, and disk usages to store working datasets. Shown in the second column, parallel FSA requires an amount of memory which is roughly proportional to the square of the dataset size. For example, to align 500 seqs it used about 23.18GB of RAM, which is expected to exceed the amount of RAM installed on common workstations. Later in this paper, we will show that memory consumption can be successfully restricted by allowing parallel FSA to interact with a database.

The third and fourth columns of Table 1 present how much database disk space is needed to store working datasets (i.e., all pairwise posterior probabilities) when indices are, and not used, respectively. For instance, to store the pairwise posterior probabilities associated with 500 seqs, about 43.30 GB of disk space is required when indices are not used. This is a reasonable size considering the current high volume and cheap prices of disks. If indices are applied on tables in the database to improve query performance, approximately 60% more disk space is expected to be required.

Table 2 presents memory consumptions and execution times with restrictions on the memory size to be consumed during the sequence annealing. The memory consumption information was obtained by parsing the stat file of the running process under the /proc directory. In the left most column, the number inside the round brackets is the desired memory size to be consumed, while a dash means that parallel FSA was run without any restriction on the memory size. In order to reduce the pairwise comparisons time, we utilized 5 processors in the cluster.

As shown in the second column of Table 2, we first note that parallel FSA used less RAM than the desired size. This feature makes it possible to align a truly large dataset with a small memory size. However, the less memory we use to align sequences, the longer execution time we need to get results, as shown in the third column. This column presents execution times of parallel FSA when indices are used for fast look-ups. Roughly speaking, aligning sequences with half of the unrestricted memory size takes three times more execution time than when the full memory size is used.

Comparison of the third and fourth columns of Table 2 shows the importance of utilizing indices. At worst, when 300 seqs were aligned with restricting the memory size to 8 GB, applying indices showed 2.6 times faster execution time than when not using the indices. When there is no restriction on memory size, the existence of indices is not important. In this case, all probability tables matching to the working dataset are transmitted from the database at once (each core running at 2.33 GHz) Intel Xeon machine with 32GB of RAM.

Table 1 contains memory consumptions for aligning different sizes of 16S datasets on a single processor, and disk usages to store working datasets. Shown in the second column, parallel FSA requires an amount of memory which is roughly proportional to the square of the dataset size. For example, to align 500 seqs it used about 23.18GB of RAM, which is expected to exceed the amount of RAM installed on common workstations. Later in this paper, we will show that memory consumption can be successfully restricted by allowing parallel FSA to interact with a database.

The third and fourth columns of Table 1 present how much database disk space is needed to store working datasets (i.e., all pairwise posterior probabilities) when indices are, and not used, respectively. For instance, to store the pairwise posterior probabilities associated with 500 seqs, about 43.30 GB of disk space is required when indices are not used. This is a reasonable size considering the current high volume and cheap prices of disks. If indices are applied on tables in the database to improve query performance, approximately 60% more disk space is expected to be required.

Table 2 presents memory consumptions and execution times with restrictions on the memory size to be consumed during the sequence annealing. The memory consumption information was obtained by parsing the stat file of the running process under the /proc directory. In the left most column, the number inside the round brackets is the desired memory size to be consumed, while a dash means that parallel FSA was run without any restriction on the memory size. In order to reduce the pairwise comparisons time, we utilized 5 processors in the cluster.

As shown in the second column of Table 2, we first note that parallel FSA used less RAM than the desired size. This feature makes it possible to align a truly large dataset with a small memory size. However, the less memory we use to align sequences, the longer execution time we need to get results, as shown in the third column. This column presents execution times of parallel FSA when indices are used for fast look-ups. Roughly speaking, aligning sequences with half of the unrestricted memory size takes three times more execution time than when the full memory size is used.

Comparison of the third and fourth columns of Table 2 shows the importance of utilizing indices. At worst, when 300 seqs were aligned with restricting the memory size to 8 GB, applying indices showed 2.6 times faster execution time than when not using the indices. When there is no restriction on memory size, the existence of indices is not important. In this case, all probability tables matching to the working dataset are transmitted from the database at once.
instead of finding and transferring relevant sets of posterior probabilities recursively.

6. Conclusion

In this work, we presented a parallel version of FSA, parallel FSA, which runs over a cluster of processors and interacts with a database while aligning the sequences. We verified performance improvements by conducting experiments over a Condor cluster. In addition to achieving speed improvements, parallel FSA has the advantage of restricting memory consumption while aligning the sequences. We believe these features of parallel FSA provide users with an ability to align truly large datasets of sequences without worry of long execution times and high memory consumption, which are current issues of existing MSA programs.

Although we used the Fixed-Size Chunking strategy as the distribution policy of sequence pairs, this strategy could possibly result in non-uniform execution times of workers when there is a wide variance in sequence length. Since the sequence annealing can only begin after getting all pairwise posterior probabilities from workers, non-uniform execution times of workers would degrade performance of parallel FSA. Therefore dynamic load-balancing of parallel FSA is a promising direction for future research.

Furthermore, we need a deep analysis of query patterns in order to pre-fetch the necessary probabilistic data from a database. Currently, during the sequence annealing, we request a query for a set of probabilistic data from a database when it is necessary, and parallel FSA is blocked until it receives the query result from the database. If sets of probabilistic data that would be necessary in future can be predicted, this information could be used to significantly improve the performance of parallel FSA.

7. Acknowledgement

We thank Colin Dewey for providing useful feedback on various parts of this work.

References


