Bioinformatics algorithm based on a parallel implementation of a machine learning approach using transducers

Abiel Roche-Lima
Department of Computer Science, E2-445 EITC. University of Manitoba, Winnipeg, Manitoba R3T 2N2, Canada

Ruppa K. Thulasiram
Department of Computer Science, E2-445 EITC. University of Manitoba, Winnipeg, Manitoba R3T 2N2, Canada, tulsi@cs.umanitoba.ca

Abstract

Finite automata, in which each transition is augmented with an output label in addition to the familiar input label, are considered finite-state transducers. Transducers have been used to analyze some fundamental issues in bioinformatics. Weighted finite-state transducers have been proposed to pairwise alignments of DNA and protein sequences; as well as to develop kernels for computational biology. Machine learning algorithms for conditional transducers have been implemented and used for DNA sequence analysis. Transducer learning algorithms are based on conditional probability computation. It is calculated by using techniques, such as pair-database creation, normalization (with Maximum-Likelihood normalization) and parameters optimization (with Expectation-Maximization - EM). These techniques are intrinsically costly for computation, even worse when are applied to bioinformatics, because the databases sizes are large. In this work, we describe a parallel implementation of an algorithm to learn conditional transducers using these techniques. The algorithm is oriented to bioinformatics applications, such as alignments, phylogenetic trees, and other genome evolution studies. Indeed, several experiences were developed using the parallel and sequential algorithm on Westgrid (specifically, on the Breeze cluster). As results, we obtain that our parallel algorithm is scalable, because execution times are reduced considerably when the data size parameter is increased. Another experience is developed by changing precision parameter. In this case, we obtain smaller execution times using the parallel algorithm. Finally, number of threads used to execute the parallel algorithm on the Breezy cluster is changed. In this last experience, we obtain as result that speedup is considerably increased when more threads are used; however there is a convergence for number of threads equal to or greater than 16.

1. Introduction and Motivation

Machine learning is concerned with the design and development of algorithms and techniques that allow computers to “learn”. Many applications of Machine Learning methods have been developed for use in Bioinformatics. Some of the most important

1 To whom any correspondence should be addressed. aroche@cs.umanitoba.ca.
approaches applied to this field are Neural Network, Support Vector Machine, Bayesian Networks and Hidden Markov Model [1-4].

On the other hand, finite automata, in which each transition is augmented with an output label in addition to the familiar input label, are considered finite-state transducers [5]. Transducer algorithms have been successfully used in a variety of applications such as speech recognition [6-8], optical character recognition [9], machine translation, a variety of other natural language processing tasks including parsing and language modeling, and image processing [10]. Indeed, transducers have been used to solve some problems in bioinformatics; weighted finite-state transducers have been proposed to pairwise alignments of DNA and protein sequences; as well as to develop kernels for computational biology [2, 11, 12]. Also, machine learning algorithms for conditional transducers have been implemented and used for DNA sequence analysis [13].

Transducer learning algorithms are based on the conditional probability computation. It is calculated using techniques, such as the creation of a pair database, normalization (with Maximum-Likelihood normalization) and parameters optimization (with Expectation-Maximization - EM) [14, 15]. All these techniques are intrinsically costly for computation when they are implemented; and when they are applied to bioinformatics the computational cost further increases. That means, it is almost unviable to use these techniques with serial implementation.

Therefore, the first objective of this work is to describe the implementation of these techniques in a bioinformatics application using parallel technologies to be utilized for conditional transducers learning. Secondly, we discuss the results when this application was executed on a grid to evaluate execution times.

WestGrid is a grid that operates across western Canada offering high performance computing, collaboration and visualization infrastructure [16]. The system software that handles the batch jobs consists of two pieces: a resource manager (TORQUE) and a scheduler (Moab). Together, TORQUE and Moab provide a suite of commands for submitting jobs, altering some of the properties of waiting jobs (such as reordering or deleting them), monitoring their progress and killing ones that are having problems or are no longer needed. In this work Westgrid is used as a baseline to test our algorithm.

The remainder of this paper is organized as follows: section 2, background and related works are described; section 3 states the specific problem; solution strategy and implementation are detailed in section 4; section 5 describes the experimental framework; finally results are illustrate in section 6, and conclusion and future works presented in section 7.

2. Background and Related Work

There have been several efforts to improve the speed and scalability of bioinformatics applications using parallel techniques. De-Cypher [17] is such a product that utilizes reconfigurable hardware to accelerate sequence analysis. Other strategies use multicore network processors for bioinformatics processing [18], as well as GPU techniques. For example, ClawHMMER [19] is a GPU-based streaming Viterbi algorithm that scales nearly
linear within a cluster of Radeon 9800 Pro GPUs. Another example is P7Viterbi algorithm described in Lindahl [20]. They used Altivec’s, including Single Instruction Multiple Data (SIMD) instructions to achieve speedup beyond Intel Pentium.

Nevertheless, there are not too many parallel algorithm related to bioinformatics applications using machine learning approaches. There are some efforts in specific implementations related to learning transducers (e.g. Maximum-Likelihood normalization, Expectation-Maximization, etc.) that has been taken into account in this work.

On the other hand, there are some bioinformatics applications running on grids. For example, on Breezy cluster (i.e. a cluster on Westgrid) there is available BLAST (Basic Local Alignment Search Tool), which is a suite of tools for assessing the similarity of a given sequence of proteins or nucleotides with a database of sequences [21].

Next section describes in details the problem we studying.

3. Specific Problem Statement

One of the main uses of the learning conditional transducer algorithm applied to bioinformatics is for classification. For example, based on a dataset with DNA sequences, classified following a criteria (i.e. …ACTGCTACTAGGGCCCTTTA… – Methanogenic; …CAGCTAAGAGCCTCTCCTTA… – Protelitic; …TAGACTACTAGGGCCCTTTA… – Cellulolitic; .TGCAAGACTAAGAGCCCTTTA… – Protelitic; …CAACTGCTACTAGGGCCCTTTA… – Methanogenic; etc.) a transducer can be learned from this dataset in order to classify future sequences. The classification could be made by calculating the stochastic edit distance based on the learned transducers.

3.1 Learning transducer algorithm

This algorithm is divided in two parts and describes in Table 1.

As can be seen in Table 1, the algorithm is based on a dataset \( (L_0) \) of classified sequences under some conditions, e.g. the sequences are classified by their functionalities as type 0 - methanogenics, type 1 - protelitics, type 2 - cellulolitic, etc. A new dataset, defined as pair-database \( (P_i) \), is created as \( (x, NN(x)) \), by calling the subalgorithm “pair-database creation”, where each sequence \( x \) is going to be paired with its nearest neighbor \( y = NN(x) \), defined as the smallest edit distance between \( x \) and \( y \) \( (d_E(x,y)) \).

The conditional transducer \( (t) \) is a matrix with the probabilities of the edit costs (insertions, deletion and substitutions) obtained by learning these values from the pair-database \( (P) \). The learning process is implemented using EM algorithm to estimate the parameters until a threshold precision is achieved.

Finally, a new sequence \( (z) \) can be classified comparing it with all elements in the original dataset \( (L_0) \) and calculating the stochastic distance using the transducer \( t \). Thus, the sequence \( (y) \) where the stochastic distance is smallest (highest conditional probability - \( p(y|z) \)) will be chosen to classify \( z \) (i.e. \( z \) will be classified as \( y \) is in \( L_0 \) : type 0 - methanogenics, type 1 - protelitics, type 2 - cellulolitic, etc.)
This algorithm is intrinsically high time-consuming. The “pair-database creation” algorithm has to create a new dataset using each element from the original dataset \((L_i)\) and look for its nearest neighbor on the same dataset. Indeed, EM algorithm uses whole pair dataset \((P_i)\) and calculates the probability to obtain a new transducer \(t\), as many times as threshold precision is achieved.

When this algorithm is used in bioinformatics dataset with several thousands of sequences, high execution times are expected. Based on our experiences, 95% of total execution time is related to “pair-database creation” and EM algorithms (approximately, “pair-database creation” 55% and EM 40%).

In the next section, we describe the solution for parallel approach and the implementation.

4. Solution Strategy and Implementation

Based on the available parallel techniques, a previous study was made to decide what method to use to tackle this problem. Initially, dependencies along the variables and data were taken into account. In the case of the “pair-database creation” algorithm, there were not considerable dependencies between the \(i\)-iteration and the \((i-1)\)-iteration. However, EM algorithm has remarkable dependencies for the different stages in the computation process.

Other considered issues to decide what parallel technique should be used were volume and communication data. In the “pair-database creation” algorithm the database size might be large, as well as communication among processors (if a distributed memory technique is chosen). In the EM algorithm will be the same, actually even worse, considering that the pair database \((P_i)\) size is twice the original dataset \((L_i)\) size. Taking these issues into consideration, a shared memory machine technique for parallel implementation was finally selected, using OpenMP (a C based parallel programming language) and C++ language.

In order to programming the general algorithm in parallel, these two high-consuming time sub-algorithms (“Pair-database creation” and EM) were re-implemented using OpenMP instructions. “Pair-database creation” algorithm uses as an input a database of sequences \((L_i)\). For each sequence \(x\) a method is called to calculate its Nearest Neighbor \((NN(x))\) in the same dataset (this algorithm order is \(O(n^2)\)). When the sequential implementation of this algorithm runs with a dataset of few thousands of sequences, the execution time is around one hour (depending of the hardware configuration). During the implementation of the parallel version, “pragma omp parallel for” instruction of OpenMP language was used, in the outer for loop of this algorithm. Initially, we got some errors regarding “Fault Segmentation”, due to “race condition”. A race condition exists when two unsynchronized threads access the shared memory to try to modify a variable at the same time. In our algorithm, it occurred when the variable used to create the pair database was accessed by different threads at the same time. This error was avoided in the implementation by including “#pragma critical” instruction, just before this variable assignation. On that way, we give exclusive right on the address space to one processor at the same time, avoiding race conditions.
The EM algorithm was also implemented in its parallelized version, using the “#pragma omp parallel for” instruction in outer for loop. In this case, dependencies are in the internal for loop, being each thread responsible for the execution of the Forward, Backward and Expectation methods involved in this algorithm. Finally, the probability is computed by summing the obtained partial sum values for each iteration. Thus, a “reduction sum” instruction was used in order to obtain the final sum of each thread.

Considering our possibility to access to Westgrid infrastructure, the parallel and sequential versions of this algorithm was executed, changing some parameters. In the next section, we describe the experiments that have been developed to evaluate this implementation.

5. Experimental Framework

Two different parameters have been changed in the algorithm to develop the experiments to compare the parallel and sequential implementation performances. These parameters are: data size and precision. Indeed, we have considered one more parameter related to the grid, i.e. the number of threads to be used during the execution of the parallel implementation.

5.1. Data size

The sequential and parallel implementations were executed for different values of the database \( L_i \) size (number of sequences). In this case, values have been increased by:

\[
\begin{align*}
\text{Database size} & = 405 & 810 & 1620 & 3240 & 6480
\end{align*}
\]

These algorithms were executed for these different values. Precision parameter was used as a constant (1E-03) as well as thread numbers (8).

5.2. Precision

By the contrary, in other set of experiments, precision values were changed to analyze execution times of these implementations. The precision parameter has an important role in the algorithm to define how many iterations have to be executed to reach the “best” transducer \( t \) and, consequently, a more precise classification. Precision parameter is used to stop the algorithm when the difference between the calculate transducer \( t \) and the one obtained in the previous iteration \( (t-1) \) is smaller than the precision value. In this case the parameter database size was kept as a constant (1620) as well as thread numbers (8), changing the precision values as follow:

\[
\begin{align*}
\text{Precision parameter} & = 1E-01 & 1E-02 & 1E-03 & 1E-04 & 1E-05
\end{align*}
\]

5.3. Numbers of Threads

Finally, a parameter related to the grid configuration was changed. It was the numbers of threads. Particularly, in this set of experiments database size was kept constant (1620), as well as precision parameters (1E-03). Thread numbers were:
When this research was started, different resources (clusters) on the Westgrid were interested to use that permits execution of OpenMP programs on a grid. Unfortunately, no other resources were open on Westgrid that allowed us to compare them.

In the next section, the main results are presented.

6. Results and Discussions

The experimental testbed, as well as the main results are described here.

6.1. Experimental testbed and environment

The experiments were developed on Breezy, a cluster of the Westgrid [16] as we have mentioned above. It is a Linux AMD stanbul cluster with large-memory nodes connected by Infiniband. Breezy is intended for jobs that need a big amount of memory on a single node, such as for large OpenMP-based parallel jobs. The Breezy specifications are summarized in Table 2.

6.2. Evaluation

On this configuration, the sequential and parallel implementations were executed, following the designed experiments.

6.2.1. Data size—Regarding the different sizes of the database, important improvement were obtained by the parallel implementation. These results can be seen in table 3 and figure 1.

The execution times and speedups were obtained for several database ($L_i$) sizes. Table 3 shows these values. They are plotted in figure 1. As can be seen in these results, when database size increases, execution times of the sequential implementation are exponentially increased. However, in the parallel implementation, execution times were lower and for larger database sizes, these values have been increased close to linear. We have also used to show the performance, the speedup metric. Speedup is defined as the ratio between the sequential and parallel execution times (i.e. $S = \frac{T_s}{T_p}$). This is a better measure that the actual execution times to set a better picture of the parallel implementation.

6.2.1. Precision—As it was mentioned above, other experiments were related to change precision parameter to obtain the learned transducer. These results can be seen in table 4 and figure 2.

As can be seen in table 4, data are collected in relation to execution times for different precision values. Indeed, speedups and reduction in execution times are computed. As it was mentioned in section 5.2, precision is used to stop the loop when the difference between $t$ and $t-1$(obtained in the last iteration) transducers is smaller than the precision parameter. For
example, after a number of repetitions (e.g. 100000) the difference between transducer $t$ and transducer $t-1$ is 0.000001. In this example the threshold value which the algorithm converges is 100000 (i.e. if the difference between $t-1$ and $t-2$ (iteration 99999) is 0.01, and the difference between $t$ and $t-1$ is 0.000001, execution times will be same value for precision parameter = 1E-03, 1E-04, and 1E-05). Thus, for precision smaller than a certain value, the number of loops will be similar, because the algorithm converges.

In figure 2 it is clear to see how the parallel algorithm starts to converge almost from the first precision value (1.00E-1). However the sequential implementation converges around the third computed precision value (1.00E-3). This tendency is due to the threshold value of the parallel implementation is reached faster, allowing the algorithm to converge from the first precision values. In other words, better solutions are achieved at much overhead implementing the algorithm in a parallel environment.

6.2.2. Numbers of Threads—Another changed parameter was the number of threads, a grid related parameter. Results are shown in figure 3.

As can be seen in this figure, data are collected in relation to execution times for different number of threads. Execution times decrease when thread numbers are added in the system, i.e. speedup is increased when the number of threads is larger. However, these nodes on Breezy cluster have 24 cores per node (see section 6.1 above), executing one thread per core. When the algorithm is executed using more than 24 threads, no result is obtained.

Indeed, for threads number equal to 16 and 24, the results are almost the same. This is because using these values, the threshold is achieved relatively close (i.e. almost at the same time). In other words, with number of thread greater than 16, the algorithm converges around nine seconds.

7. Conclusions and Future Work

In this paper, we have described the problem, approach and evaluation of a parallelized implementation of the conditional transducer learning algorithm applied to bioinformatics.

The most time consuming sub-algorithms (95% of the total time) were parallelized using OpenMP. These are “pair-database creation” and EM. Several experiments were designed to evaluate our implementation, using as a baseline Breezy cluster, in Westgrid grid. Experiments were grouped in three different goals: (1) increasing database size, (2) increasing precision, and (3) changing the numbers of threads.

Obtained results have permitted to analyze this application on the grid. When database sizes were increased, execution time values of the sequential algorithm rose in an exponential trend. However, these values for parallel implementation were increased close to linear.

On the other hand, important results were obtained when parallel implementation was executed on the grid changing the precision parameter. Execution time values of the parallel algorithm were lower than sequential implementation. Indeed, the algorithm converged for
the first precision values in the parallel proposal, while it was achieved later (for larger precision values) in the sequential approach.

Another important result was related to the number of threads used to execute the parallel implementation of the algorithm on the Westgrid. Considering the configuration of this cluster, the maximum number of threads available for nodes is 24. When the number of threads was increased, the speedup rose. However, there is a convergence in the execution times when the number of threads is equal to or greater than 16.

Concluding, the parallel implementation was able to guarantee considerably low execution times when it was executed on the Breezy cluster. It permits to speed up the performance of the algorithm.

In future work, some changes will be implemented in the algorithm to use larger sequences. When these modifications are made, a new parallelized implementation will be programmed and evaluated.

References

[1]. Hua SJ, Sun ZR. A novel method of protein secondary structure prediction with high segment overlap measure: Support vector machine approach. Journal of Molecular Biology. 2001; 308(2): 397–407. [PubMed: 11327775]
[2]. Durbin, R.; Eddy, SR.; Krogh, A., et al. Biological sequence analysis: Probabilistic Models of Proteins and Nucleic Acids. Cambridge University Press; 1998.
[3]. Soding J. Protein homology detection by HMM-HMM comparison (vol 21, pg 951, 2005). Bioinformatics. 2005; 21(9):2144–2144.
[4]. Pollastri G, Przybylski D, Rost B, et al. Improving the prediction of protein secondary structure in three and eight classes using recurrent neural networks and profiles. Proteins-Structure Function and Genetics. 2002; 47(2):228–235.
[5]. Berstel, J. Transductions and Context-Free Languages. Teubner Studienbucher; Stuttgart: 1979.
[6]. Allauzen, C.; Mohri, M. Linear-Space Computation of the Edit-Distance between a String and a Finite Automaton. College Publications; 2009.
[7]. Mohri, M. Statistical natural language processing. Cambridge University Press; 2005.
[8]. Mohri, M.; Pereira, FCN.; Riley, M.; Rabiner, Larry; Juang, Fred. Handbook on speech processing and speech communication, Part E: Speech recognition. Springer-Verlag; Heidelberg, Germany: 2008. Speech recognition with weighted finite-state transducers.
[9]. Breuel TM. The OCROPUS open source OCR system. Document Recognition and Retrieval Xv. 2008; 6815
[10]. Albert, J.; Kari, J. Digital image compression. In: Droste, Manfred; Kuich, Werner; Vogler, Heiko, editors. Handbook of weighted automata. EATCS Monographs on Theoretical Computer Science; 2009.
[11]. Allauzen, C.; Mohri, M.; Talwalkar, A. Sequence kernels for predicting protein essentiality.
[12]. Pevzner, PA. Computational Molecular Biology: an Algorithmic Approach. MIT Press; 2000.
[13]. Roche-Lima, A.; Grave de Peralta, RA.; Cora, MA., et al. Bioinformatics applied to genetic study of rumen microorganism. 2007.
[14]. Oncina J, Sebban M. Learning stochastic edit distance: Application in handwritten character recognition. Pattern Recognition. 2006; 39(9):1575–1587.
[15]. Bernard M, Janodet JC, Sebban M. A discriminative model of stochastic edit distance in the form of a conditional transducer. Grammatical Inference: Algorithms and Applications, Proceedings. 2006; 4201:240–252.
[16]. WestGrid | Western Canada Research Grid. http://www.westgrid.ca/
[17]. TimeLogic biocomputing products from Active Motif, Inc. http://www.timelogic.com/
23-12-2010

[18]. Wun, B.; Buhler, J.; Crowley, P. Exploiting coarsegrained parallelism to accelerate protein motif
finding with a network processor; PACT ’05: Proceedings of the 2005 International Conference
on Parallel Architectures and Compilation Techniques; 2005.

[19]. Horn, DR.; Houston, M.; Hanrahan, P. Clawhammer: A streaming hmmer-search implementation;
SC ’05: The International Conference on High Performance Computing, Networking and
Storage; 2005.

[20]. Lindahl, E. Altivec-accelerated HMM algorithms. 2008. http://lindahl.sbc.su.se/

[21]. BLAST - Basic Local Alignment Search Tool | WestGrid. http://www.westgrid.ca/support/
software/blast
Figure 1.
Results for different values of database size for sequential and parallel implementation
Figure 2.
Results for different values of precision for sequential and parallel implementation
Figure 3.
Results for different values of numbers of threads
Table 1
General idea of the Learning Conditional Transducer Algorithm

| Part I. Learning a stochastic transducer |
|------------------------------------------|
| Step 1: The dataset will be considered the Learning Set (\(L_i\)), i.e. each sequence is classified under some consideration (i.e. type 0 - "methanogens", type 1 - "proteolitics", type 2 - "celulolitic", etc.). |
| Step 2: from each sequence in \(L_i\), a set of string pairs (pair dataset) \(P_i\) will be built in the form \((x, y)\), where \(y = \text{NN}(x) = \text{argmin}_{y \in L_i} d_E(x, y)\) \((y = \text{NN}(x)\) is the Nearest Neighbor of \(x\), calculated as the minimum value \((\text{argmin})\) of the \(d_E\) (the classic edit distance) between \(x\) and all other values in the same dataset \((y \in L_i)\). |
| Step 3: a unique conditional transducer \(t\) will be learned from all pair dataset \((\bigcup \ P_i)\) |

| Part II. Classification process |
|--------------------------------|
| For a new given sequence \(z\), |
| Step 4: \(z\) will be classified as the same type as \(y\), where \(y \in L_i\) maximizing \(p(y|z)\), where \(p(y|z)\) is the conditional probability calculated using the learned transducer \(t\). |
Table 2
Specifications of the Breezy cluster on the Westgrid

| Processors:                          |
|--------------------------------------|
| ■ Quad-socket, 6-core AMD Istanbul processors (24 cores @ 2.4 GHz) per node |
| ■ 256 GB memory per compute node (64 GB on login node)                        |
| ■ 16 nodes connected with 4X DDR InfiniBand at 20 Gbits/s and Gigabit Ethernet |

| Interconnect:                       |
|-------------------------------------|
| ■ 6 nodes connected with 4X DDR InfiniBand at 20 Gbits/s and Gigabit Ethernet |

| Storage:                            |
|-------------------------------------|
| ■ 10 TB total for /global/scratch   |
| ■ 10 TB total for /home             |
Table 3
Results of the sequential and parallel implementation for different data sizes

| Database size (\# sequences) | Execution Time (sec) | Speedup S = T_p/T_s | Reduction in execution time (%) when parallelized |
|------------------------------|----------------------|---------------------|-----------------------------------------------|
|                              | Parallel Implementation | Sequential Implementation |                                               |
| 405                          | 17.8                 | 66.9                | 3.8                                           | 73.7                                          |
| 810                          | 34.0                 | 188.6               | 5.6                                           | 82.1                                          |
| 1620                         | 90.0                 | 567.3               | 6.3                                           | 84.1                                          |
| 3240                         | 290.2                | 1857.1              | 6.4                                           | 84.4                                          |
| 6480                         | 816.4                | 5307.0              | 6.5                                           | 84.6                                          |
Table 4

Results of the sequential and parallel implementation for different precision values

| Precision | Execution Time (sec) | Speedup | Reduction in execution time (%) when parallelized |
|-----------|----------------------|---------|--------------------------------------------------|
|           | Parallel Implementation | Sequential Implementation |       |                                                   |
| 1E-01     | 14.9                 | 68.0    | 4.5     | 77.7                                              |
| 1E-02     | 17.3                 | 97.2    | 5.6     | 82.1                                              |
| 1E-03     | 18.4                 | 118.0   | 6.3     | 84.1                                              |
| 1E-04     | 18.8                 | 116.8   | 6.2     | 83.8                                              |
| 1E-05     | 19.3                 | 121.6   | 6.3     | 84.1                                              |