A Parallel Approach for Evolutionary Induced Decision Trees. MPI+OpenMP Implementation

Marcin Czajkowski(✉), Krzysztof Jurczuk, and Marek Kretowski

Faculty of Computer Science, Bialystok University of Technology,
Wiejska 45a, 15-351 Bialystok, Poland
{m.czajkowski,k.jurczuk,m.kretowski}@pb.edu.pl

Abstract. One of the important and still not fully addressed issues in evolving decision trees is the induction time, especially for large datasets. In this paper, the authors propose a parallel implementation for Global Decision Tree system that combines shared memory (OpenMP) and message passing (MPI) paradigms to improve the speed of evolutionary induction of decision tree. The proposed solution is based on the classical master-slave model. The population is evenly distributed to available nodes and cores, and the time consuming operations like fitness evaluation and genetic operators are executed in parallel on slaves. Only the selection is performed on the master node. Efficiency and scalability of the proposed implementation is validated experimentally on artificial datasets. It shows noticeable speedup and possibility to efficiently process large datasets.

Keywords: Evolutionary algorithms · Decision trees · Parallel computing · MPI · OpenMP

1 Introduction

Evolutionary algorithms (EA) [14] are metaheuristic nature-inspired algorithms that represent techniques for solving a wide variety of difficult optimization problems. Their mechanisms such as mutation, recombination, natural selection and survival of the fittest are inspired by the biological evolution. One of the main drawbacks of EA is relatively high computational complexity. This issue is especially important for current data mining applications [6], where larger and larger datasets are processed and analyzed.

Fortunately EA are naturally prone to parallelism and the process of artificial evolution can be implemented in various ways [1]. It is possible to parallelize time consuming operations or to parallelize the whole evolutionary process itself. The first approach is often based on the master-slave model [7] and aims at speeding up the calculation without changing the original sequential algorithm. The second approach leads to a variety of distributed (coarse-grained) and cellular (fine-grained) algorithms that differ from the sequential implementation.

In the recent past evolutionary algorithms were successfully applied to evolve decision trees as an alternative to the greedy top-down approaches [2]. However,

L. Rutkowski et al. (Eds.): ICAISC 2015, Part I, LNAI 9119, pp. 340–349, 2015.
DOI: 10.1007/978-3-319-19324-3_31

evolving decision trees is usually more costly and time-consuming and considerably limits their popularity comparing to the greedy strategies. In the recent survey [2] on the evolutionary induction of decision trees, authors put on the first place in the future trends the need of speeding up the evolutionary tree induction.

In this paper, the authors investigate how the evolutionary induction of decision tree can be parallelized using both shared address space (OpenMP [3]) and message passing (MPI [16]) paradigms on a cluster of nodes with multi-core chips. The main objectives of this work are to accelerate the Global Decision Tree (GDT) system and to allow efficient evolutionary induction of decision trees on large datasets.

The first attempt to parallelize the GDT solution was proposed in [13]. The authors investigated parallel and distributed solutions for global induction of decision trees. In this paper, we rewrite and significantly extend that parallel implementation and introduce hybrid MPI+OpenMP approach that may provide a better efficiency than e.g. pure MPI version [15]. In addition, the proposed solution focuses not only on the algorithm speedup but also on algorithm's ability to efficiently process large datasets.

This paper is organized as follows. The next section provides a brief background on the GDT system. Section 3 describes our approach for parallel implementation of evolutionary tree induction in detail. Section 4 presents experimental validation of the proposed solution and comparison results on artificial datasets. In the last section, the paper is concluded and possible future works are sketched.

2 Global Decision Tree System

The GDT general structure follows a typical framework of evolutionary algorithms [14] with an unstructured population and a generational selection. It is able to induce univariate [10], oblique [11] and mixed [12] classification trees.

Decision trees are complicated tree structures, in which number of nodes, type of the tests and even number of test outcomes are not known in advance. Therefore, in the GDT system individuals are not specially encoded and are represented in their actual form as a typical classification trees. Depending on the tree type (univariate, oblique, mixed), each test in internal node concerns one or more attributes. In case of univariate tests, a test representation depends on the considered attribute type. For nominal attributes at least one attribute value is associated with each branch starting in the node, which means that an internal disjunction is implemented. Typical inequality tests with two outcomes are used for continuous-valued features. Only precalculated candidate thresholds [5] are considered as potential splits. In an oblique test with binary outcome a splitting hyperplane is represented by a fixed-size table of real values corresponding to a weight vector and a threshold. The inner product is calculated to decide where an example is routed.

Initial individuals are created by applying the simple top-down algorithm based on a dipolar principle [9] to randomly selected sub-samples of the learning

set [12]. Ranking linear selection [14] is used as a selection mechanism. Additionally, in each iteration a single individual with the highest value of fitness function in current population is copied to the next one (elitist strategy). Evolution terminates when the maximum number of generations (default value: 1000) is reached.

To maintain genetic diversity, two specialized genetic operators corresponding to the classical mutation and cross-over were proposed. They are applied with a given probability to a tree (default value is 0.8 for mutation and 0.2 for cross-over). Mutation operator starts with randomly choosing the type of node (equal probability to select leaf or internal node). Next, the ranked list of nodes of the selected type is created and a mechanism analogous to the ranking linear selection is applied to decide which node will be affected. Depending on the type of node, the ranking takes into account the location of the internal node (internal nodes in lower parts of the tree are mutated with higher probability) and the number of misclassified objects (nodes with worse classification accuracy are mutated with higher probability). Modifications performed by the mutation operator depend on the tree type and the node type (i.e. if the considered node is a leaf node or an internal node) and cover different variants:

- changing the sub-trees or tests in the internal nodes;
- pruning the internal nodes or expanding the leaves that contain objects from different classes.

Cross-over operator starts with selecting positions in two affected individuals. Depending on the recombination variant, randomly selected nodes may:

- exchange subtrees (if exists);
- exchange tests associated with the nodes (only when non-terminal nodes are chosen and the numbers of outcomes are equal);
- exchange branches in random order which start from the selected nodes (only when non-terminal nodes are chosen and the numbers of outcomes are equal).

Successful application of any operator results in a necessity for relocation of the learning examples between tree parts rooted in the modified nodes.

Fitness function is one of the most important and sensitive factor in the design of EA. It drives the evolutionary search process by measuring how good a single individual is in terms of meeting the problem objective. In context of decision trees a direct minimization of the reclassification quality measured on a learning set usually leads to the overfitting problem. This problem is partially mitigated by defining a stopping condition and by applying a post-pruning [4] in typical top-down induction of decision trees [17]. In case of the evolutionary induced decision trees, this problem may be mitigated by a term incorporated into the fitness function. In the GDT system the fitness function is maximized and has the following form:

$$Fitness(T) = Q_{Reclass}(T) - \alpha \cdot Comp(T),$$

where $Q_{Reclass}(T)$ is the reclassification quality of the tree T and α is the relative importance of the classifier complexity (default value is 0.005). The tree

complexity term $Comp(T)$ can be viewed as a penalty for over-parametrization. It includes the tree size (calculated as the number of leaves) and for oblique and mixed trees also the complexity of attributes in the internal nodes.

3 Parallel Implementation of Global Decision Tree System

In this section, the parallel implementation of the GDT system is proposed. At first, an efficient fitness calculation is shortly discussed and next, distributed (MPI) and shared (OpenMP) memory solutions are described.

In a typical EA the evaluation of fitness of individuals in population is the most time consuming operation. As it is calculated independently for every individual, this process can be easily parallelized by distributing population evenly among available nodes (slaves). The master node executes the remaining operations of the evolution.

In case of the GDT system, the aforementioned approach cannot be directly applied. The information about the learning vectors is stored in each node of decision trees. This way the genetic operators can efficiently and directly obtain the fitness corresponding to the individual [12], [8]. The actual fitness calculation is embedded into the post mutation and cross-over processing, when the learning vectors in the affected parts of the tree (or trees) are relocated. This mechanism increases the memory complexity of the induction but significantly reduces its computational complexity. As a consequence, the most time consuming elements of the algorithm are genetic operators and they should be performed in parallel.

Figure 1 illustrates the proposed hybrid parallel approach for the evolutionary induced decision tree algorithm. It can be observed that at the first step the master node spreads individuals from the population over slave nodes using message-passing strategy. In the next step, in each slave node the calculations are spread over cores which run the algorithm blocks in parallel.

It should be recalled that the shared memory approach is strongly linked and limited by the available hardware (e.g. 8 cores in one node), whereas within the distributed memory approach it is usually easier to create more numerous configurations.

3.1 Distributed Memory Approach

In each evolutionary loop, the master evenly distributes individuals between the nodes (slaves). To avoid wasting resources, the chunk of population is left on the master which also works as a slave. Migration the individuals between nodes is performed with the framework of the message-passing interface and requires:

- packing the tree structures into a flat message;
- transfer the message between nodes (sending/receiving);
- unpacking the message into the corresponding tree.

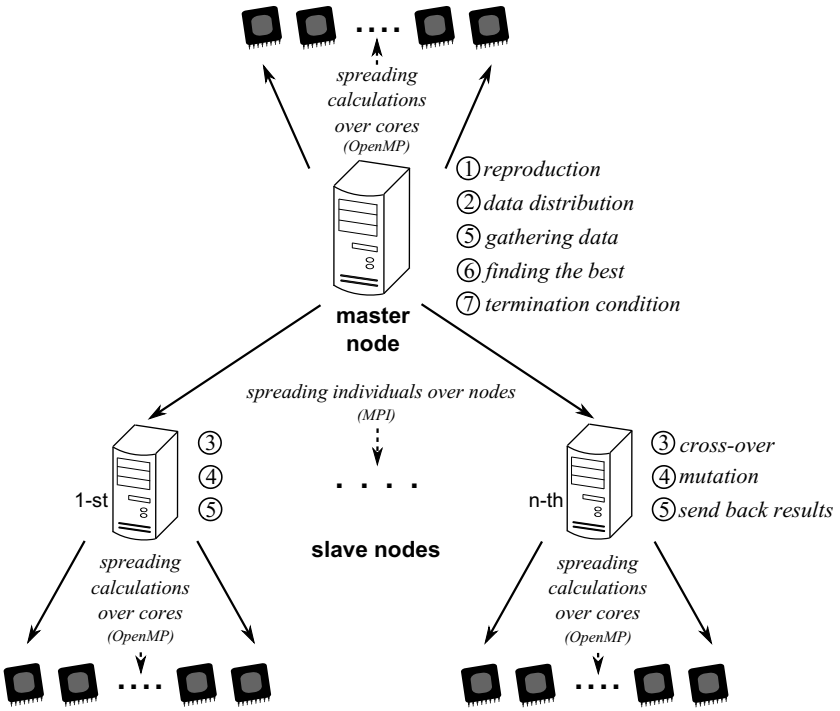


Fig. 1. Hybrid parallel approach of the evolutionary induced decision tree algorithm

The packed tree structure contains information about its size, the tests in the internal nodes' and additional nodes statistics (e.g. number of learning vectors), that speeds up the reconstruction during the message unpacking in the target node. In order to minimize the message size, the information about learning vectors associated with the tree nodes is not included in the message.

The certain parts of EA like reproduction (with elitism) and terminal condition verification are executed on the master node. However, to perform the selection, the fitness value of the distributed individuals has to be known. To avoid unnecessary unpacking-packing operations (for trees will not be selected into the next generation) on the master, the fitness value of the migrated individual is also transferred. Additionally, a certain number of individuals from the given slave node may survive (or be replicated) and in the next iteration they could be scheduled to be sent back to that node. This observation gives another possibility to eliminate unproductive calculation. However, it raises a risk that the individuals in particular a node may not change much (or be very similar). To keep the original sequential algorithm and avoid some kind of island models, we sent the cloned trees to random slave nodes to keep the sub-population diversified at each slave and to avoid crossing with identical or very similar individuals.

In the previously presented research [13] on the parallel GDT implementation, the redistribution of learning vectors was performed after unpacking each

individual on the target node. The whole tree was reconstructed before starting the mutation and cross-over operations. Then, after successful application of a genetic operator, the redistribution of learning vectors of an affected node (and eventual sub-nodes) was performed.

This process of associating each instance with appropriate leaf is very time-consuming, especially on large datasets. In order to limit redistribution of the data, we propose to reconstruct only those nodes that will be affected. In addition, there is only a need to fit learning vectors that fall to the affected node since the redistribution of eventual sub-nodes is not necessary. If a genetic operator will be successful, the learning vectors in the sub-tree will be relocated anyway. This way, instead of reallocating all learning vectors in whole tree, we only set a part of the data in the node (without its eventual sub-nodes) that is selected for mutation or cross-over. It can be also noticed that if the root tree node is to be affected by a genetic operator, the preceding processing is reduced only to associating the whole dataset to the root node. The GDT assumption that internal nodes in lower parts of the tree are mutated with higher probability also enhances a possible speedup of the proposed implementation as it is expected that the lower parts of the tree held fewer learning vectors that need to be assigned.

3.2 Shared Memory Approach

The shared memory approach is applied in every slave node (including master which works also as a slave). We assume that all cores within the node operate independently but share the same memory resources. Access and modification of the same memory space by one core is visible to all other cores, therefore, no data communication between the cores is required. However, additional synchronization during write/read operations is needed in order to insure appropriate access to shared memory.

In Figure 1 we see that each slave node spreads calculations further. The calculations in the chosen algorithm blocks concerning different individuals are spread over cores. This way, all variants of genetic operation together with redistribution of learning vectors can be performed in parallel. In case of mutation, each core processes a single individual at a time, whereas during cross-over, pairs of affected individuals are processed in parallel. Parallelization with shared memory approach is also applied on the master node for the distribution and gathering population from other nodes. In addition, all trees that were transformed into leaves after application of genetic operators are extended into sub-trees in parallel by cores at each slave node.

4 Experiments

In this section we show the performance of the proposed parallel version of the GDT system. Two sets of experiments were performed. At first, the efficiency of the parallel MPI and OpenMP implementation is presented for four datasets.

Next, more detailed information is illustrated for one selected dataset with respect to speedup and size of the dataset.

4.1 Setup

Experimental verification was performed with the mixed version of the GDT system. All presented results were obtained with a default settings of parameters from the sequential version of the GDT system. We have tested four artificially generated datasets with different characteristics described in Table 1 and illustrated in Figure 2. All datasets are composed of 100 000 instances and have different characteristics like number of attributes and classes or the type of optimal splits.

In the paper we focus only on the time performance of the GDT system, therefore, results for the classification accuracy are not enclosed. However, for all tested datasets, the GDT system managed to induce trees with optimal structures and almost perfect accuracies (99%-100%). For detailed accuracy results, we refer reader to our previous papers [10,12].

In the performed experiments a cluster of sixteen SMP servers (nodes) running Ubuntu 12 and connected by an Infiniband network (20 Gb/s) was used. Each server was equipped with 16GB RAM, 2xXeon X5355 2.66GHz CPUs with total number of cores equal 8. We used the Intel version 15.1 compiler, MVAPICH version 2.2 and OpenMP version 3.0. Within each node, only the shared memory approach (OpenMP) was applied whereas between the nodes the message-passing interface (MPI) was used.

4.2 Results

In the first experiment, the authors focus on the overall speedup of the proposed hybrid MPI-OpenMP approach. Table 2 presents the obtained mean speedup for different datasets (100 000 instances). Only the best combination of nodes and cores is shown and it looked as follows for all four datasets:

- results for 2 cores: 1 node with 2 OpenMP threads;
- results for 4 cores: 1 node with 4 OpenMP threads;
- results for 8, 16, 32, 64 cores: 8 nodes with 1, 2, 4, 8 OpenMP threads per node, respectively.

Table 1. Datasets' characteristics: name, number of instances, number of attributes, number of classes and the type of splits in the internal nodes

Dataset	Instances	Attributes	Classes	Splits
Chess 3x3	100 000	2	2	univariate
Cross	100 000	2	5	univariate
Diamond	100 000	2	2	oblique
Zebra	100 000	10	2	oblique

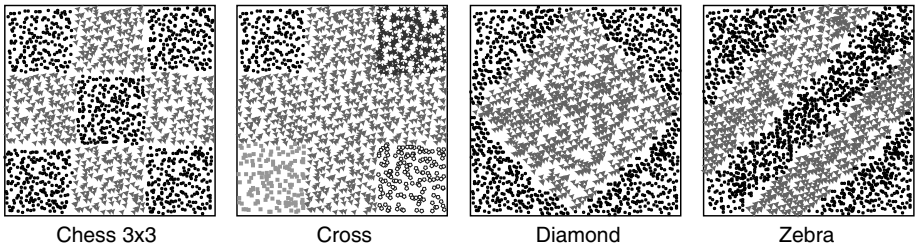


Fig. 2. Examples of artificial datasets

Table 2. Mean speedup reported for different number of cores

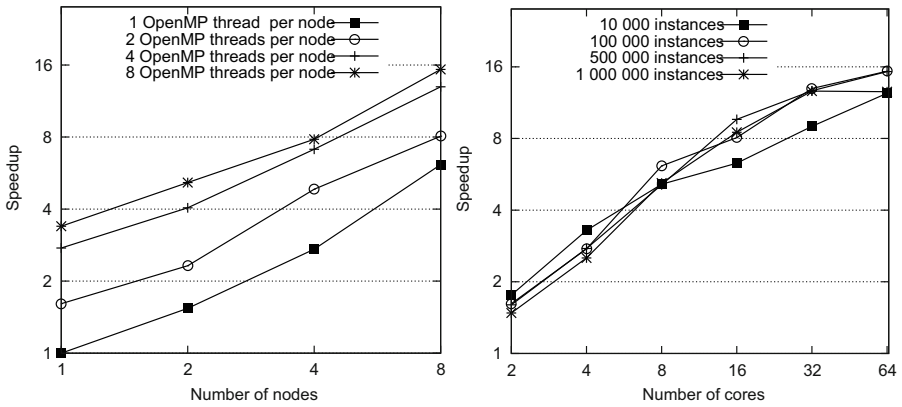
Dataset	Speedup on different number of cores					
	2	4	8	16	32	64
Chess 3x3	1.62	2.75	6.14	8.06	12.97	15.34
Cross	1.67	2.61	4.90	7.80	9.69	10.59
Diamond	1.86	2.62	4.61	7.10	9.35	10.58
Zebra	1.66	2.61	4.43	5.60	8.91	9.93

It is clearly visible that the hybrid parallel algorithm is able to noticeable decrease the computation time. The best speedup for 64 cores (8 cluster nodes - 1 MPI process per node and 8 OpenMP threads inside each node) is obtained for the dataset *Chess 3x3*. We can observe that the speedup differences between 32 and 64 cores are relatively small considering doubling the number of cores. One of the reasons is the size of the population (default: 64 individuals). To achieve effective parallelization, the number of cores should not exceed half of the population because for some operations like cross-over, each core performs calculations on two individuals. The second reason why the efficiency for the higher number of cores is getting smaller results from the Amdahl's law [7] as some parts of the algorithm have to run sequentially.

It should be noticed, that in previous work [13] speedup of the parallel implementation of GDT system on similar datasets were between 2 and 3 for 8 processing units. Here, we manage to achieve speedups between 4 and 6.

Fig. 3(a) shows how the number of used OpenMP threads per node influences the simulation time. Results are obtained for the dataset *Chess 3x3* (100 000 instances). Each time all cores on the used nodes are allocated. Although the population size equals 64, it is still profitable to use 8 nodes (8 MPI processes) with 8 OpenMP threads per each node.

The detailed results for different dataset (*Chess 3x3*) size are presented in Fig. 3(b). It shows the optimal results for the different number of used cores. For example, the 16 used cores means: i) for 10 000 instances - 4 cluster nodes - 1 MPI process per node and 4 OpenMP threads inside each node, or ii) for 1



a) b)

Fig. 3. Performance of the hybrid parallel algorithm of evolutionary induced decision tree: a) speedup across nodes with different number of OpenMP threads per node, b) speedup for different dataset size

000 000 instances - 8 cluster nodes - 1 MPI process per node and 2 OpenMP threads inside each node. Each time all cores on the used nodes are allocated. It is visible that the algorithm deals successfully both with small and large dataset sizes. It is true for a few cores as well as for more processing units.

Time comparison of the average loop time of the GDT solution shows that it is scalable in context of the data size. The sequential GDT system performs evolutionary loop in an average 0.037 second for *Chess 3x3* dataset with 10 000 instances and 31.71 seconds for 1 million instances. The default number of generations in the GDT system equals 1 000, therefore, the tree induction time for the sequential algorithm and the largest dataset takes almost 9 hours (+ the time to read the dataset and create initial population) whereas using parallel implementation is reduced up to 42 minutes when 64 cores are used.

5 Conclusion and Future Works

In the paper, the hybrid parallelization of the evolutionary induction of decision tree is investigated. The authors manage to successfully speed up the evolutionary induction of decision trees and efficiently process large datasets. Proposed implementation takes an advantage of modern parallel machines and may provide an efficient acceleration on high-performance computing clusters as well as on low-cost commodity hardware.

We will continue to work with the presented approach to adapt it to evolutionary induction of regression and model trees. Moreover, future work will deal with a GPGPU paralellization.

Acknowledgments. This project was funded by the Polish National Science Center and allocated on the basis of decision 2013/09/N/ST6/04083. The second and third author was supported by the grant S/WI/2/2013 from Bialystok University of Technology.

References

1. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6(5), 443–462 (2002)
2. Barros, R.C., Basgalupp, M.P., Carvalho, A.C., Freitas, A.A.: A survey of evolutionary algorithms for decision-tree induction. IEEE Transactions on Systems Man and Cybernetics Part C Applications and Reviews 42(3), 291–312 (2012)
3. Chapman, B., Jost, B.G., Pas, R., van der Kuck, D.J.: Using OpenMP: Portable Shared Memory Parallel Programming. MIT Press (2007)
4. Esposito, F., Malerba, D., Semeraro, G.: A comparative analysis of methods for pruning decision trees. IEEE Transactions on Pattern Analysis and Machine Intelligence 19(5), 476–491 (1997)
5. Fayyad, U., Irani, K.: Multi-interval discretization of continuous-valued attributes for classification learning. In: Proc. of IJCAI, pp. 1022–1027 (1993)
6. Freitas, A.: Data mining and knowledge discovery with evolutionary algorithms. Natural Computing Series. Springer (2002)
7. Grama, A., Karypis, G., Kumar, V., Gupta, A.: Introduction to Parallel Computing. Addison-Wesley (2003)
8. Kalles, D., Papagelis, A.: Lossless fitness inheritance in genetic algorithms for decision trees. Soft Computing 14(9), 973–993 (2010)
9. Krętowski, M.: An evolutionary algorithm for oblique decision tree induction. In: Rutkowski, L., Siekmann, J.H., Tadeusiewicz, R., Zadeh, L.A. (eds.) ICAISC 2004. LNCS (LNAI), vol. 3070, pp. 432–437. Springer, Heidelberg (2004)
10. Kretowski, M., Grześ, M.: Global learning of decision trees by an evolutionary algorithm. In: Information Processing and Security Systems, pp. 401–410 (2005)
11. Krętowski, M., Grześ, M.: Evolutionary learning of linear trees with embedded feature selection. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Żurada, J.M. (eds.) ICAISC 2006. LNCS (LNAI), vol. 4029, pp. 400–409. Springer, Heidelberg (2006)
12. Kretowski, M., Grześ, M.: Evolutionary induction of mixed decision trees. International Journal of Data Warehousing and Mining 3(4), 68–82 (2007)
13. Krętowski, M., Popczyński, P.: Global induction of decision trees: from parallel implementation to distributed evolution. In: Rutkowski, L., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2008. LNCS (LNAI), vol. 5097, pp. 426–437. Springer, Heidelberg (2008)
14. Michalewicz, Z.: Genetic algorithms + data structures = evolution programs, 3rd edn. Springer (1996)
15. Rabenseifner, R., Hager, G., Jost, G.: Hybrid MPI/OpenMP parallel programming on clusters of multi-core SMP nodes. In: Proc. of the 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing, pp. 427–436 (2009)
16. Pacheco, P.: Parallel Programming with MPI. Morgan Kaufmann Publishers (1997)
17. Rokach, L., Maimon, O.Z.: Data mining with decision trees: theory and application. Machine Perception Artificial Intelligence 69 (2008)