GPU-Accelerated Evolutionary Induction of Regression Trees

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

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

Abstract. In the paper, we investigate the speeding up of the evolutionary induction of decision trees, which is an emerging alternative to greedy top-down solutions. In particular, we design and implement graphics processing units (GPU)-based parallelization to generate regression trees (decision trees employed to solve regression problems) on large-scale data. The most time consuming part of the algorithm, which is parallelized, is the evaluation of individuals in the population. Other parts of the algorithms (like selection, genetic operators) are performed sequentially on a CPU. A data-parallel approach is applied to split the dataset over the GPU cores. After each assigned chunk of data is processed, the results calculated on all GPU cores are merged and sent to the CPU. We use a Compute Unified Device Architecture (CUDA) programming model, which supports general purpose computation on a GPU (GPGPU). Experimental validation of the proposed approach is performed on artificial and real-life datasets. A computational performance comparison with the traditional CPU version shows that GPU-accelerated evolutionary induction of regression trees is significantly (even up to 1000 times) faster and allows for processing of much larger datasets.

Keywords: Evolutionary Algorithms · Decision trees
Parallel computing · Graphics processing unit (GPU) · Regression trees
Large-scale data

1 Introduction

Evolutionary Algorithms (EAs) [21] are naturally prone to parallelism. The artificial evolution process can be parallelized using various strategies [6] and different implementation platforms [13]. Recently, GPGPU has been widely used in EAs parallelization due to its high computational power at a relatively low cost [2]. It allows us to reduce the CPU load on the most time-consuming operations. The paper covers the parallelization of the evolutionary induction of decision trees (DT)s [18], which represents one of the major and frequently applied techniques for discriminant analysis prediction in data mining [12]. Traditionally, DTs are induced with greedy top-down strategy, however, in the recent past an evolutionary approach for the tree induction has attracted a great deal of interest. The evolutionary induced DTs [3] are much simpler than the ones generated by a greedy strategy [24] with at least comparable prediction performance. The main downside of the evolutionary approach is the relatively higher computational costs due to EA itself. Thus, evolutionary induction of DTs using large-scale data become very time-demanding.

In this paper, we focus on speeding up the evolutionary induction of regression trees that are considered as a variant of DTs, designed to approximate real-valued functions instead of being used for classification tasks [5]. The proposed GPU parallelization handles the most computing intensive jobs like fitness calculation, leaving the evolutionary flow control and communication to the CPU. It is applied to a framework called Global Decision Tree (GDT) that can be used for evolutionary induction of classification [19] and regression [9] trees. The manuscript can be seen as a continuation of previous study on GPU-based approach to evolutionary induced classification trees [16]. It extends the research to regression trees which demand a more advanced parallelization schema (e.g. for predictions calculation in the leaves, dipole mechanism) to evaluate and evolve individuals.

The paper is organized as follows. The next section provides a brief background. Section 3 describes our approach for GPU-accelerated evolutionary induction of regression trees. The experimental evaluation is performed in Sect. 4 on artificial and real-life datasets. In the last section, the paper is concluded and possible future works are outlined.

2 Background

In this section, we present some background information on evolutionary induced regression trees, GPGPU computing model and recent related works.

2.1 Evolutionary Induced Regression Trees

There are different variants of DTs in the literature [10]. They can be grouped according to the type of problem they are applied to and the way they are induced. In classification trees, a class label is assigned to each leaf. Regression trees may be considered as a variant of decision trees designed to approximate real-valued functions instead of being used for classification tasks. In a basic variant of a regression tree, each leaf contains a constant value, usually equal to an average value of the target attribute of all training instances that reach that particular leaf. To predict the value of the target attribute, the new tested instance is followed down the tree from a root node to a leaf using its attribute values to make routing decisions at each internal node. Next, the predicted value for the new instance is evaluated based on prediction associated with the leaf. Although regression trees are not as popular as classification ones, they are highly competitive with different machine learning algorithms [23].

Traditionally, DTs are induced with a greedy procedure known as recursive partitioning [24]. In this top-down approach the induction algorithm starts from a root node where the locally optimal split of the data is found according to the given optimality measure. Next, the training instances are redirected to newly created nodes, and this process is repeated for each node until a stopping condition is met. Additionally, post-pruning is usually applied after the induction to avoid the problem of over-fitting the training data.

An alternative concept for the decision tree induction focuses on a global approach which limits the negative effects of locally optimal decisions. It tries to simultaneously search for the tree structure and the tests in the internal nodes. This process is obviously much more computationally complex but can reveal hidden regularities that are often undetectable by greedy methods. Global induction is mainly represented by systems based on an evolutionary approach [3]. In the literature, there are relatively fewer evolutionary approaches for the regression trees than for the classification ones. Popular representatives of EA-based regression trees are the TARGET solution [11] that evolves a CART–like regression tree with basic genetic operators and a strongly typed genetic programming approach called STGP [15].

2.2 GPGPU

A general-purpose computation on GPUs (GPGPU) stands for the use of graphics hardware for generic problems. One of the most popular frameworks to facilitate GPGPU is a Compute Unified Device Architecture (CUDA) [27] created by the NVIDIA Corporation. In the CUDA programming model, a GPU is considered as a co-processor that can execute thousands of threads in parallel to handle the tasks traditionally performed by the CPU. This CPU load reduction using GPGPU is recently widely applied in many computational intelligence methods [26]. Application of GPUs in evolutionary data mining usually focuses on boosting the performance of the evolutionary process which is relatively slow due to high computational complexity, especially for the large scale data [2].

When the CPU delegates a job to the GPU, it calls a kernel that is a function run on the device. Then, a grid of (threads) blocks is created and each thread executes the same kernel code in parallel. The GPU computing engine is an array of streaming multiprocessors (SMs). Each SM consists of a collection of simple streaming processors (called CUDA cores). Each block of threads is mapped to one of the SMs, and the threads inside the block are mapped to CUDA cores.

There are two decomposition techniques that are most commonly used to parallelize EAs [1]: a data approach and a control approach. The first decomposition strategy, which is also applied in this paper, focuses on splitting the dataset and distributing its chunks across the processors of the parallel system. The second approach focuses on population decomposition as individuals from the population are evaluated at the same time on different cores [14]. The main drawback of this approach is relatively weak scalability for large-scale data.

2.3 Related Works

Speeding up the DT induction has so far been discussed mainly in terms of classification problems. In the literature, we may find some attempts at parallelization the tree building process, however, most of the implementations focus on either a greedy approach [20] or random forests [25]. Despite the fact that there is a strong need for parallelizing the evolutionary induced DT [3], the topic has not yet been adequately explored. One of the reasons is that the straightforward application of GPGPU to EA may be insufficient. In order to achieve high speedup and exploit the full potential of such parallelization, there is a need to incorporate knowledge about DT specifically and its evolutionary induction.

In one of the few papers that cover parallelization of evolutionary induced DT, a hybrid MPI+OpenMP approach is investigated for both classification [7] and regression [8] trees. The algorithms use the master-slave paradigm, and the most time-consuming operations, such as fitness evaluation and genetic operators, are executed in parallel on slaves nodes. The authors apply the control parallelization approach in which the population is evenly distributed to the available nodes and cores. The experimental validation shows that the possible speedup of such a hybrid parallelization is up to 15 times for 64 CPU cores.

To the best of our knowledge, in the literature there is one study that covers GPGPU parallelization of evolutionary induced DTs [16]. Experimental validation on artificial and real-life datasets showed that it was capable of inducing trees two orders of magnitude faster in comparison to the traditional CPU version. However, it concerned only classification trees.

3 GPU-Accelerated Induction of Regression Trees

In this section, we briefly describe the original evolutionary tree induction and next, we propose an efficient acceleration of it using GPGPU.

3.1 Global Decision Tree Induction Framework

The general structure of the GDT system follows a typical EA framework [21] with an unstructured population and a generational selection. GDT allows evolving different kinds of tree representations [10], however; in our description we focus on univariate trees in which each split in the internal node is based on a single attribute. Individuals are represented in their actual form as regression trees and initialized using a simple top-down greedy algorithm on a random subsample of the training data. The tests in the internal nodes are found on random subsets of attributes.

Tree-based representation requires developing specialized genetic operators corresponding to classical mutation and crossover. The GDT framework [9] offers several specialized variants that can modify simultaneously the tree structure and tests in internal nodes. The mutation operator makes random changes in nodes of the selected individuals by e.g. replacing the test, shifting its threshold, pruning the non-terminal nodes or expanding the leaves. To construct a new test in the internal node GDT uses a locally optimized strategy called 'long dipole' [9]. At first, an instance that will constitute the dipole is randomly selected from the set of instances from a current node. The rest of the objects are sorted in decreasing order according to the difference between the dependent variable values and the selected instance. Next, the second instance that constitutes the dipole with possibly a much different dependent variable value is searched for using a mechanism similar to the ranking linear selection [21]. Finally, the test that splits the dipole is constructed based on a randomly selected attribute. The threshold value is randomly selected from a range defined by the pairs that constitute the dipole.

The crossover operator attempts to combine elements of two existing individuals (parents) to create a new solution. Randomly selected nodes may exchange the tests, branches or even the subtrees in asymmetrical manner [9]. Both operators are applied with a given probability to a tree (default value is 0.8 for mutation and 0.2 for crossover). Selecting the point of mutation or crossover depends on the location (level) of the node in the tree and its average prediction error per instance. This way the weak nodes (with high error value) and the ones from the lower parts of the tree are selected with higher probability. Successful application of any operator results in the necessity for relocation of the learning instances between tree parts rooted in the modified nodes. In addition, in every node, information about training instances currently associated with the node is stored. This makes it faster to perform local structure and test modifications during applications of genetic operators. However, it increases the memory consumption.

Fitness function is one of the most important and sensitive elements in the design of EAs. It drives the evolutionary search process by measuring how good a single individual is in terms of meeting the problem objective. GDT framework offers different multi-objective strategies like weight formula, lexicographic analysis or Pareto dominance. Here, we use the first strategy and apply the following expression for the fitness function:

$$Fitness(T) = [1 - 1/(1 + RMSE(T))] + \alpha(S(T) - 1.0), \tag{1}$$

where S is the tree size expressed as a number of nodes, $RMSE$s is root-mean-square error, α is the relative importance of the complexity term and a user supplied parameter (default value is 0.0005).

The selection mechanism is based on a ranking linear selection [21] with the elitist strategy, which copies the best individual founded so far to the next population. Evolution terminates when the fitness of the best individual in the population does not improve during a fixed number of generations (default: 1000) or maximum number of generations is reached (default: 5000).

3.2 GPU-Based Approach

The proposed algorithm is based on a data decomposition strategy. In this approach, each GPU thread operates on a small fraction of the dataset. The general flowchart of our GPU-based approach is illustrated in Fig. 1. It can be seen that only the most time consuming operation of EA, the evaluation of the individuals, is performed in parallel on GPU. The parallelization does not affect the behavior of the original EA as the evolutionary induction flow is driven by the CPU in a sequential manner.

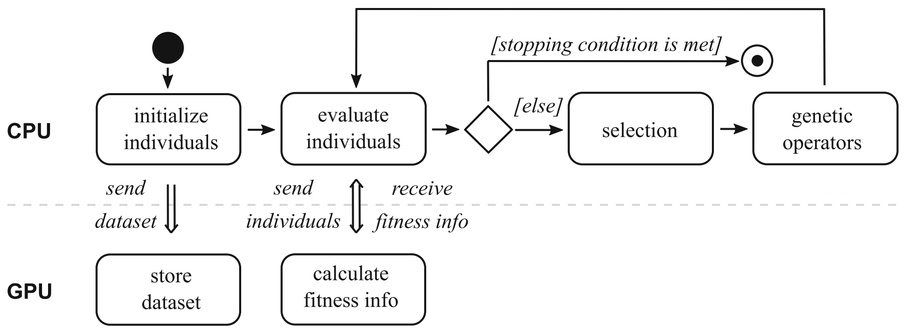

Fig. 1. Flowchart of a GPU-accelerated algorithm.

The first modification of the GDT framework concerns the initialization phase that begins by sending and saving the whole dataset from CPU to the global GPU memory. This way the GPU threads have constant access to the data and the heaviest data transfer is performed only once. The following operations: initialization of the population as well as selection of the individuals remain unchanged compared to original GDT system. The reason why these initial steps are not parallelized is that the initial population is created only once on small fractions of the dataset. In the evolutionary loop, CPU is also involved in relatively fast operations like genetic operators and selection. After successfully application of crossover or mutation, there is a need to evaluate the individuals. For calculating $RMSE$ and fitness, all objects in the training dataset need to be passed through the tree starting from the root node to an appropriate leaf. As this is a time-consuming operation and can be performed in parallel, it is delegated to the GPU which performs all necessary objects relocations and fitness calculation.

The cooperation between CPU and GPU is organized in four kernel calls that can be grouped into two sets: $Kernel1_{pre/post}$ and $Kernel2_{pre/post}$ (Fig. 2). They cover the decomposition phase (*pre*) and gathering phase (*post*) which is illustrated in Fig. 2. The role of the first function named $Kernel1_{pre}$ is to propagate objects from the root of the tree to the leaves. Each GPU block makes a copy of the evaluated individual that is later loaded into the shared memory that is visible in all threads within the block. The dataset is spread into smaller parts, first between different GPU blocks and then further between the threads. This way the threads can process the same individual but perform calculations on different chunks of the data. In each tree leaf the sum of predictions for the

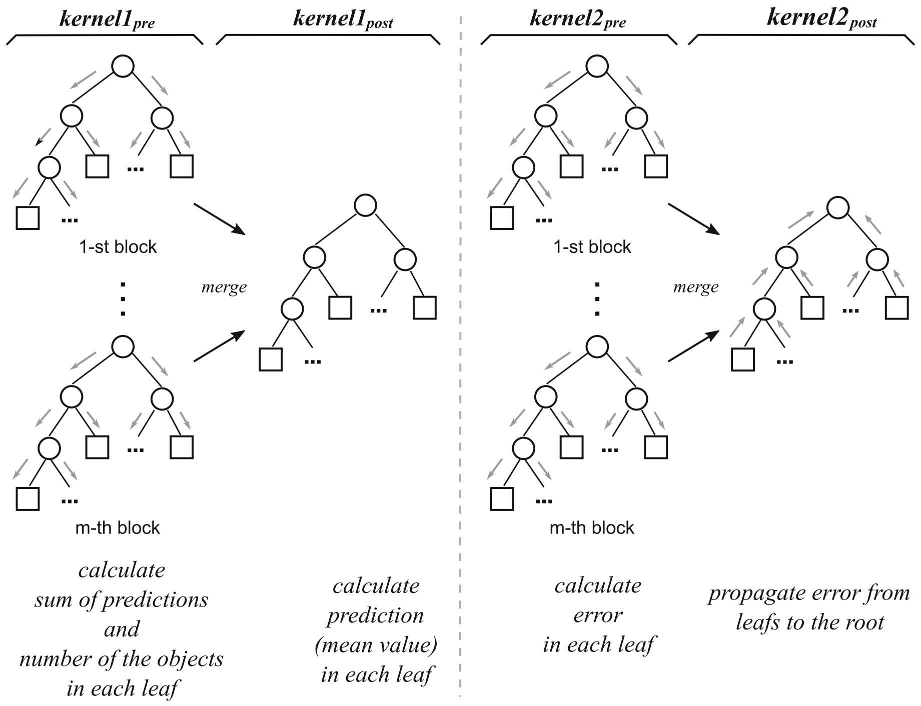
kernel1pre
kernel1post
kernel2pre
kernel2post
1-st block
m-th block
merge
merge
1-st block
m-th block
calculate sum of predictions and number of the objects in each leaf
calculate prediction (mean value) in each leaf
calculate error in each leaf
propagate error from leafs to the root

Fig. 2. Four kernel functions responsible finally for fitness info calculation for each individual.

training instances that reach that particular leaf as well as the number of the objects are stored. Next, the gathering function $Kernel1_{post}$ merges information from multiple copies of the individual allocated in each GPU block (see Fig. 2). For each individual, information that was calculated by the threads and stored in the leaves is combined.

The role of the $Kernel2$ functions is to enable the fitness calculation of the individual. In order to do that, the tree error must be determined using the information calculated by the $Kernel1$ functions and stored in the leaves. $Kernel2_{pre}$ function again splits the dataset into small parts and next in each GPU block its threads propagate and assign objects to the tree leaves. This operation is necessary to calculate the squared error of each tree leaf in $Kernel2_{pre}$ function and finally overall sum of squared residuals in the $Kernel2_{post}$ call. Like in $Kernel1_{pre}$ function, each GPU block stores a copy of the individual. The threads within the blocks calculate the prediction (mean value) for every leaf of every individual in the population from the information gathered in $Kernel1$ function. Next, each thread sums the squared differences between assigned objects predictions and the leaf prediction. The $Kernel2_{post}$ function gathers and merges the information of the squared error for each leaf from GPU blocks, determines sum of squared residuals and propagates the tree error from the leaves to the root node.

To improve the algorithm's performance, during the evolutionary induction CPU does not have access to the objects that fall into particular nodes of the tree as the propagation of the instances is performed on GPU. However, some variants of the mutation operator, that searches for the 'long dipole', require at least two objects to construct a new test in the internal node. That is why, in our GPU-accelerated approach each tree node contains additional information about two instances that may constitute 'long dipole'. First instance is randomly selected in the $Kernel1_{pre}$ function and the second one is set during gathering phase in the $Kernel2_{post}$ function. When the multiple copies of the tree are merged, in each node the second instance is searched from available set of instances in other GPU blocks according to the differences in the dependent variable value. As both instances are selected randomly and should have much different target values, the general concept of the 'long dipole' used in CPU version is maintained.

4 Experimental Validation

In this section, the performance analysis of the GPU-accelerated algorithm is verified, both on large-scale artificial and real-life datasets. As we are focused in this paper only on speeding up the GDT system, the results for the prediction performance are not included. For detailed information about the GDT prediction performance please see our previous papers [9,10].

4.1 Setup

In all experiments a default set of parameters from the sequential version of the GDT system is used and the results correspond to averages of 10 runs. We have tested two artificially generated datasets called *armchair* and *chess* (1, 5, 10 and 20 millions of instances, 2 real-valued attributes) [7,8] and two large real-life publicly available datasets: *Suzy* (5 millions of instances, 17 real-valued attributes) and $Year$ (515 345 instances, 90 real-valued attributes) available in the UCI Machine Learning Repository [4]. Due to the lack of publicly available large-scale regression datasets, *Suzy* which originally concerned classification was transformed in such a way that in performed experiments the value of the last attribute is predicted instead of the class label. The only purpose of this operation was to investigate the algorithm's speedup and not the prediction performance.

All the experiments were performed on a regular PC equipped with a processor Intel Xeon E5-2620 v4 (20 MB Cache, 2.10 GHz), 64 GB RAM, and a single graphics card. We used a 64-bit Ubuntu Linux 16.04.02 LTS as an operating system. The sequential algorithm was implemented in C++ and compiled with the use of gcc version 5.4.0. The GPU-based parallelization was implemented in CUDA-C and compiled by nvcc CUDA 8.0 [22] (single-precision arithmetic was applied). We tested three NVIDIA GeForce GTX graphics cards:

– 780 (2304 CUDA cores, clock rate 863 MHz, 3 GB of memory with 288.4 GB/s bandwidth),
– Titan Black (2880 CUDA cores, clock rate 889 MHz, 6 GB of memory with 336.0 GB/s bandwidth),
– Titan X (3072 CUDA cores, clock rate 1000 MHz, 12 GB of memory with 336.5 GB/s bandwidth).

4.2 Results

Table 1 shows the obtained speedup of the proposed GPU-accelerated algorithm in comparison to its sequential version. Speedup for three GPUs and different datasets are included. It is clearly visible that the proposed GPU-acceleration provides a significant decrease in computation time. NVIDIA GTX Titan X GPU is able to speed up the evolutionary induction even more than $\times 1000$, while other GPUs allow us to decrease the computation time at least $\times 100$.

The scale of the improvement is even more visible when comparing the execution time between the sequential and parallel version of the GDT system (Table 2). For large data, the tree induction time for the proposed solution can be counted in minutes, while the original sequential algorithm often needs at least a few days. Moreover, the achieved speedup is much higher than the one obtained by a computer cluster of 16 nodes each equipped with 2 quad-core CPUs (Xeon 2.66 GHz) (128 CPU cores in total) and 16 GB RAM [8].

Table 1. Mean speedup for different datasets and various GPUs.

Dataset	GTX 780	GTX Titan Black	GTX Titan X
Armchair1M	×470	×496	×1189
Armchair5M	×572	×535	×1328
Armchair10M	×529	×546	×1372
Armchair20M	×505	×458	×1349
Chess1M	×140	×121	×232
Chess5M	×176	×192	×300
Chess10M	×181	×188	×249
Chess20M	×163	×129	×259
Year	×421	×512	×885
Suzy	×344	×324	×760

The results suggest that with the proposed approach even a regular PC with a medium-class graphics card is enough to significantly accelerate the GDT tree induction time. As it is expected, better graphics cards manage to achieve much better accelerations. However, the NVIDIA GTX 780 GPU is much cheaper (about 10 times) than NVIDIA GTX Titan X GPU and it provides only 2 or 3 times lower speedup.

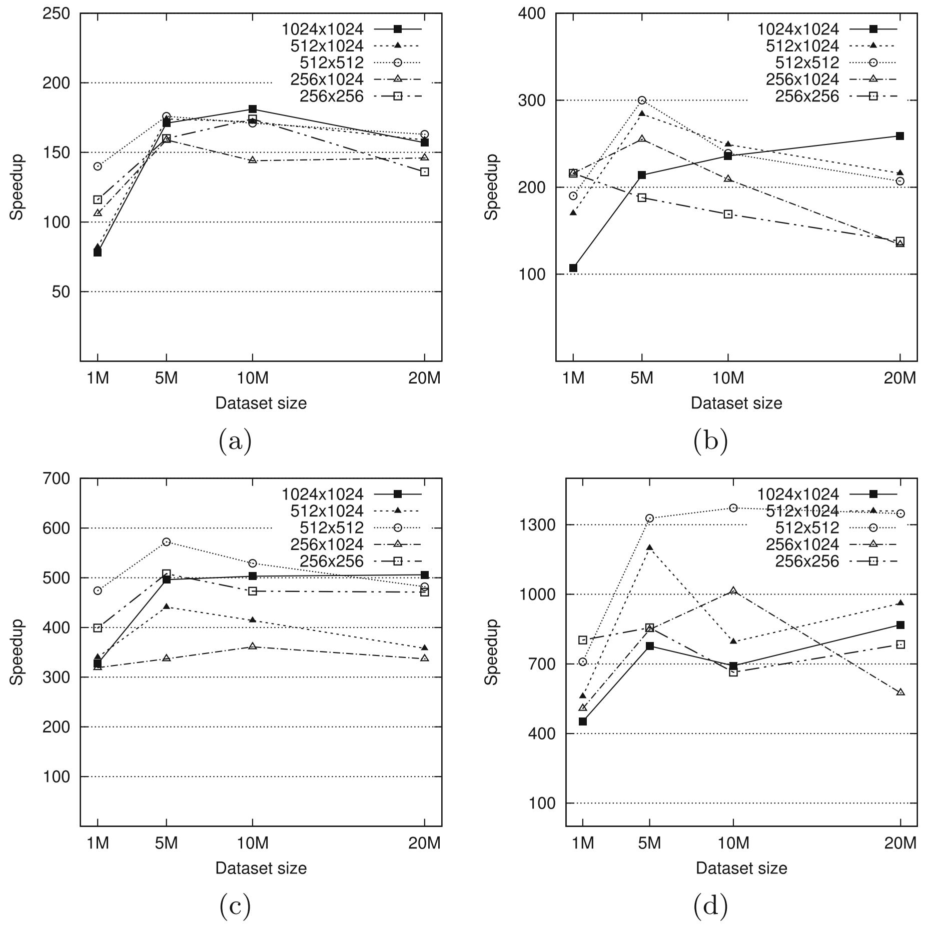

Fig. 3. The mean speedup for a few blocks × threads configurations and the different size of the dataset (1, 5, 10, and 20 million instances), successively for: (a) Geforce GTX 780, *Chess*, (b) Geforce GTX Titan X, *Chess*, (c) Geforce GTX 780, *Armchair*, (d) Geforce GTX Titan X, *Armchair*.

There is also a difference in speedup between datasets and/or their size. The highest acceleration was when using the *Armchair* dataset. This dataset is the simplest one and the induced trees are the smallest. This can suggest that the size of the generated regression trees (problem difficulty) influences time performance due to CUDA thread/branch divergence [27]. More investigation is needed here, e.g. detailed time profiling of both CPU and GPU.

We also experimentally verified whether different sizes of the data processed in each block/thread influences the algorithm time. For this purpose, we tested various blocks × threads configurations using two datasets (*Chess* and *Armchair*) and the slowest/quickest GPU (GTX 780 and GTX Titan X). Figure 3

Table 2. Mean execution time of the sequential algorithm as well as its GPU-accelerated version on the fastest GPU (in seconds and days/hours/minutes).

Dataset	Sequential	GPU-accelerated
Armchair1M	74 783 s ≈ 21 h	63 s ≈ 1 min
Armchair5M	404 388 s ≈ 4.5 days	304.5 s ≈ 5 min
Armchair10M	781 668 s ≈ 9 days	570 s ≈ 9.5 min
Armchair20M	1 492 440 s ≈ 17 days	1106 s ≈ 18.5 min
Chess1M	102 599 s ≈ 1 days 4 h	442 s ≈ 7 min
Chess5M	550 915 s ≈ 6 days 9 h	1 834 s ≈ 30.5 min
Chess10M	1 050 515 s ≈ 12 days	4 219 s ≈ 1 h 10.5 min
Chess20M	2 076 507 s ≈ 24 days	8 020 s ≈ 2 h 13.5 min
Year	90 360 s ≈ 25 h	102 s ≈ 1.5 min
Suzy	710 000 s ≈ 8 days 5 h	934 s ≈ 15 min

shows that for all larger datasets (starting with 10M), the configuration with more blocks/threads fits the best, whereas for smaller datasets configurations with less blocks/threads gives noticeably better results. There are at least two reasons that may explain the described algorithm behavior. Too small data portions per thread could cause more overhead as there are more threads to create, manage, and so on. On the other hand, the problem with load balancing can exist when the chunks of data are too big.

5 Conclusion

This paper focuses on GPU-accelerated evolutionary induction of regression trees. Proposed implementation takes an advantage of the specificity of evolutionary DT induction to exploit the full potential of GPGPU approach. Presented results show that our solution is fast, scalable and can easily explore large-scale data.

We see many promising directions for future research. In particular, we are currently working on extending our approach on regression trees with linear models in the leaves (model trees) and multi-GPU platforms. There are also many interesting ideas for optimization the proposed algorithm like processing only the modified by the genetic operators part of the tree instead of propagation all dataset objects. We also plan to verify the influence of various GPU specific memory improvements [17] in order to speed up the algorithm further.

Acknowledgments. This work was supported by the grants S/WI/2/13 (first and third author) and W/WI/1/2017 (second author) from Bialystok University of Technology founded by Ministry of Science and Higher Education.

References

1. Alba, E., Tomassini, M.: Parallelism and evolutionary algorithms. IEEE Trans. Evol. Comput. **6**(5), 443–462 (2002)
2. Bacardit, J., Llor, X.: Large-scale data mining using genetics-based machine learning. WIRE Data Min. Knowl. Discov. **3**(1), 37–61 (2013)
3. Barros, R.C., Basgalupp, M.P., Carvalho, A.C., Freitas, A.A.: A survey of evolutionary algorithms for decision-tree induction. IEEE Trans. SMC Part C **42**(3), 291–312 (2012)
4. Blake, C., Keogh, E., Merz, C.: UCI repository of machine learning databases (1998). http://www.ics.uci.edu/~mlearn/MLRepository.html
5. Breiman, L., Friedman, J.H., Olshen, R.A., Stone, C.J.: Classification and Regression Trees. Wadsworth and Brooks, Monterey (1984)
6. Chitty, D.: Fast parallel genetic programming: multi-core CPU versus many-core GPU. Soft Comput. **16**(10), 1795–1814 (2012)
7. Czajkowski, M., Jurczuk, K., Kretowski, M.: A parallel approach for evolutionary induced decision trees. MPI+OpenMP implementation. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2015. LNCS (LNAI), vol. 9119, pp. 340–349. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-19324-3_31
8. Czajkowski, M., Jurczuk, K., Kretowski, M.: Hybrid parallelization of evolutionary model tree induction. In: Rutkowski, L., Korytkowski, M., Scherer, R., Tadeusiewicz, R., Zadeh, L.A., Zurada, J.M. (eds.) ICAISC 2016. LNCS (LNAI), vol. 9692, pp. 370–379. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-39378-0_32
9. Czajkowski, M., Kretowski, M.: Evolutionary induction of global model trees with specialized operators and memetic extensions. Inf. Sci. **288**, 153–173 (2014)
10. Czajkowski, M., Kretowski, M.: The role of decision tree representation in regression problems an evolutionary perspective. Appl. Soft Comput. **48**, 458–475 (2016)
11. Fan, G., Gray, J.B.: Regression tree analysis using TARGET. J. Comput. Graph. Stat. **14**(1), 206–218 (2005)
12. Fayyad, U.M., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: Advances in Knowledge Discovery and Data Mining. AAAI Press, Menlo Park (1996)
13. Gong, Y.J., Chen, W.N., Zhan, Z.H., Zhang, J., Li, Y., Zhang, Q., Li, J.J.: Distributed evolutionary algorithms and their models: a survey of the state-of-the-art. Appl. Soft Comput. **34**, 286–300 (2015)
14. Grama, A., Karypis, G., Kumar, V., Gupta, A.: Introduction to Parallel Computing. Addison-Wesley, Boston (2003)
15. Hazan, A., Ramirez, R., Maestre, E., Perez, A., Pertusa, A.: Modelling expressive performance: a regression tree approach based on strongly typed genetic programming. In: Rothlauf, F., et al. (eds.) EvoWorkshops 2006. LNCS, vol. 3907, pp. 676–687. Springer, Heidelberg (2006). https://doi.org/10.1007/11732242_64
16. Jurczuk, K., Czajkowski, M., Kretowski, M.: Evolutionary induction of a decision tree for large-scale data: a GPU-based approach. Soft Comput. (2017, in press)
17. Jurczuk, K., Kretowski, M., BezyWendling, J.: GPU-based computational modeling of magnetic resonance imaging of vascular structures. Int. J. High Perform. Comput. Appl. (2017, in press)
18. Kotsiantis, S.B.: Decision trees: a recent overview. Artif. Intell. Rev. **39**(4), 261–283 (2013)

19. Kretowski, M., Grześ, M.: Global learning of decision trees by an evolutionary algorithm. In: Saeed, K., Pejaś, J. (eds.) Information Processing and Security Systems, pp. 401–410. Springer, Boston (2005). https://doi.org/10.1007/0-387-26325-X_36
20. Lo, W., Chang, Y., Sheu, R., Chiu, C., Yuan, S.: CUDT: a CUDA based decision tree algorithm. Sci. World J. 1–12 (2014)
21. Michalewicz, Z.: Genetic Algorithms + Data Structures = Evolution Programs, 3rd edn. Springer, London (1996). https://doi.org/10.1007/978-3-662-03315-9
22. NVIDIA: CUDA C programming guide. Technical report (2017). https://docs.nvidia.com/cuda/cuda-c-programming-guide/
23. Ortuno, F.M., Valenzuela, O., Prieto, B., Saez-Lara, M.J., Torres, C., Pomares, H., Rojas, I.: Comparing different machine learning and mathematical regression models to evaluate multiple sequence alignments. Neurocomputing **164**, 123–136 (2015)
24. Rokach, L., Maimon, O.: Top-down induction of decision trees classifiers - a survey. IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.) **35**(4), 476–487 (2005)
25. Strnad, D., Nerat, A.: Parallel construction of classification trees on a GPU. Concurr. Comput. Pract. Exp. **28**(5), 1417–1436 (2016)
26. Tsutsui, S., Collet, P.: Massively Parallel Evolutionary Computation on GPGPUs. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-37959-8
27. Wilt, N.: CUDA Handbook: A Comprehensive Guide to GPU Programming. Addison-Wesley, Boston (2013)