Combined adaptive load balancing algorithm for parallel applications

Luka Filipović, Božo Krstajić, *Member IEEE*, Tomo Popović, *Senior Member IEEE*

***Abstract*— Development and improvement of efficient techniques for parallel task scheduling on multiple cores processors is one of the key issues encountered in parallel and distributed computer systems. The purpose of process distribution improvement in parallel applications is in increased system performance, reduced application execution time, reduced losses and increased resource utilization.**

This paper presents combined adaptive load balancing algorithm based on domain decomposition and master-slave algorithms and its core scheduling adaptive mechanism that handles load redistribution according obtained and analyzed data. Selection of distribution algorithm, based on collected parameters and previously defined conditions, proved to deliver increased performances and reduced imbalance. Results of simulations confirm better performance of proposed algorithms compared to the standard algorithms reviewed in this paper.

***Index Terms*— parallel programming, load balancing algorithm, tash scheduling, adaptive algorithm.**

I. INTRODUCTION

Distributed computer systems enables the delivery of computing resources necessary to solve complex problems with requirements that exceed the capabilities of the most powerful personal computers. High-performance computers, as one of the powerful elements of distributed computer systems, lead to complex solutions by using computer simulations enabling progress in all scientific fields. Parallel processing supports execution of several processes and instructions simultaneously, with a goal to save time and execute faster and more efficient complex applications in scientific and industrial applications. [1] [2].

The focus of many researches in the parallel processing field is process of finding optimal distribution of tasks in order to increase efficiency, reduce execution time of parallel applications and reduce communication time of computer resources. In order to achieve the highest parallel application efficiency, it is crucial to optimize the assignment of tasks to parts of the distributed computer system (cluster nodes and its CPU cores) and monitor their execution.

The subject of this research was combined adaptive algorithm (CAA) [3][4], which uses combination the static and dynamic load balancing algorithms to improve the performance of independent parallel tasks scheduling without significantly complicating the whole process. It uses an adaptive innovative mechanism for choosing load balancing algorithm for distribution of unexecuted autonomous tasks depending on the segments in which losses are the least and by limiting the algorithm at times when it causes losses.

II. LOAD BALANCING ALGORITHMS

Load balancing in parallel processing is defined as process of achieving parallelism by redistributing the load of parallel segments during the execution of a parallel program [5] [6]. The primary goal of load balancing algorithms is to find the optimal execution schedule that defines the initial execution time and the execution order of all tasks that run on a particular resource. Load balancing of parallel applications is process of reducing computation time achieved by reducing communication time, synchronization time between processes and waiting time due to uneven process distribution [7].

The imbalance of parallel applications most often occurs due to uneven load between cores, excessive communication between cores or waiting of group of cores for others to finish assigned jobs [8]. In a real distributed environment, resource load varies over time and it is not always possible to improve the use of resources that are completely free or equally loaded. It is not possible to determine or predict the length of processes that run on separate computers or delays due to communication between computers. Therefore, there is a longer execution of the parallel application and a decrease in resource utilization. The end of the execution of a parallel application or the beginning of the postprocessing phase directly depends on the execution time of the part of the application on the core that is assigned the most process or the processor with the lowest frequency.

Load balancing algorithms are divided as static and dynamic, depending on the type of job scheduling. Static load balancing algorithms have good usability and efficiency on homogeneous clusters while they execute tasks on all cores which have similar duration. Performance of programs using these algorithms is reduced at the end of the runtime without possibility of rescheduling. One of widely used static algorithms is domain decomposition algorithm. On the other side, dynamic algorithms can give better efficiency on heterogeneous system, but make unnecessary communication during executing time. The master slave algorithm is a one of the typical representatives of dynamic algorithms. Domain decomposition and master-slave algorithms have their advantages and disadvantages depending on the characteristics of the resource, the specific parallel application for which load balancing is performed and the duration of processes that are executed in parallel [9-11].

Luka Filipović is with the University Donja Gorica, Oktoih 2, Podgorica, Montenegro (e-mail: luka.filipovic@udg.edu.me).

Božo Krstajić is with the Faculty of electrical engineering, University of Montenegro, Džordža Vašingtona bb, Podgorica, Montenegro (e-mail: bozok@ucg.ac.me).

Tomo Popović is with the Faculty of Information Systems and Technologies, University Donja Gorica, Oktoih 2, Podgorica, Montenegro (e-mail: tomo.popovic@udg.edu.me).

Adaptive algorithms are advanced dynamic algorithms with adaptive strategy for task distribution scheme that is activated depending on the load change of the distributed system during operation.

III. Combined Adaptive Load Balancing Algorithm

The combined adaptive algorithm (CAA) is successor an improved version of combined algorithm (CA) [12]. It presents an adaptive decision model that selects an adequate algorithm based on data on the state of the resource on which the parallel application is running and the duration of finished tasks.

In the preprocessing phase, as in the CA algorithm, the input data is divided and tasks are prepared for execution. Before starting parallel simulations, the analysis of the distributed resource configuration is performed and the obtained data are used in the later analysis.

In the parallel processing of the combined adaptive load balancing algorithm, three execution phases stand out (Figure 1):

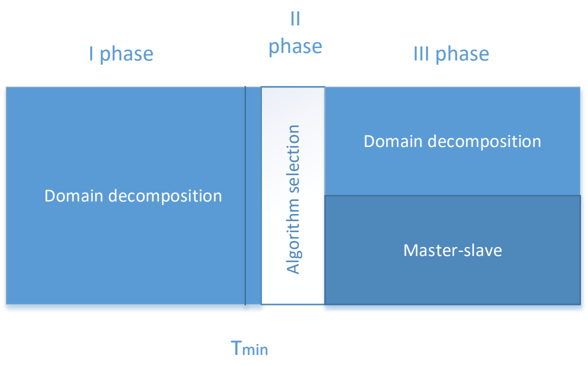

Fig 1. Execution phases of the proposed CAA algorithm

In the first phase of the combined adaptive algorithm, the domain decomposition algorithm is executed. It has the highest efficiency and the lowest losses in the initial phase of program execution. The algorithm stops working when the first (“fastest”) core completes the assigned job (T_{min}) and sends instructions to the other cores to stop working after completing the task they are processing at that point. The described procedure reduces the losses of the first execution phase to a minimum.

In the second phase of the algorithm, based on the amount and duration of performed tasks, cluster configuration and its load, an adaptive approach is used to select the algorithm for the scheduling of the remaining tasks in third phase. Upon initiating an interrupt at the end of the first phase, each CPU core sends to a predefined core a data containing the duration of the performed tasks. The predefined core receives the sent data and processes them, making an array with the number of executed tasks for each core and through executed and unexecuted tasks and selects the algorithm to be executed in the third phase according to the defined decision algorithm. The decision on the algorithm in third phase is made on the basis of the following parameters:

- the homogeneity of allocated resources,
- the total number of assigned cores,
- the numbers of completed tasks for each core individually and
- the execution time of each task individually.

The homogeneity of the allocated resources (the examination of whether they can be considered homogeneous or heterogeneous) is performed by comparing the performance values of the allocated nodes of the distributed resources. A measure of the performance of an individual resource can be core frequency, node memory or node network speed. Depending on the architecture of the distributed system and the type of tasks, one or more node performance measures can be taken. In the presented research, the core frequency (Hz) was used as a measure of the node performance of the distributed system.

The total number of assigned cores is defined when the application is started.

The number of completed tasks per core represents the part of the total number of tasks performed up to the moment T_{min}, when the first core performed the assigned tasks and initiated the interrupt, for each core separately. The data is expressed as a sequence whose number of elements is equal to the number of assigned cores, and the elements are the numbers of completed tasks for each core individually. The total number and type of tasks depends on the parallel application being executed and the input data, and the division is done before the parallel processing.

The execution time of each individual task is a matrix that contains the data on which core the task was executed and the duration of each task (ms) that was completed.

Based on the above parameters, the conditions for selection of an adequate distribution algorithm in the third phase can be defined. These conditions are defined by variables U_i that have binary values. Thus, the variable U_i takes the value 1 if the i-th condition is met, and otherwise U_i takes the value 0.

The first and eliminatory condition (U_e) for the selection of the distribution algorithm is the condition that the remaining number of tasks is less than or equal to the number of available cores. If the conditions U_e ($U_e = 1$) are met, the DD algorithm is selected for execution in the third phase, ie each of the remaining tasks is assigned one core for execution.

If the eliminator condition is not met ($U_e = 0$), the choice of algorithm is made based on a combination of the following conditions:

- U_1 - cluster homogeneity condition: this condition is fulfilled (U_1=1) if CPU cores of the same or approximate operating clock are assigned, ie. if the standard deviation of the operating clock of all cores is less than the set value;
- U_2 - number of cores condition: this condition is fulfilled ($U_2 = 1$) if the number of cores is less than a predefined number of cores, ie if the losses of the master core in the MS algorithm cannot be ignored;
- U_3 - condition of uniformity of the number of performed tasks: this condition is fulfilled ($U_3 = 1$) if the number of performed tasks for each core is approximate, ie. if the value of the standard deviation of the number of completed tasks per core is less than the predetermined value;
- U_4 - condition of uniformity of duration of performed tasks: this condition is fulfilled ($U_4 = 1$) if the duration of performed tasks per core is approximate, ie. the

value of the standard deviation of the execution time of each task per core is less than the predefined value.

The decision algorithm checks the fulfillment of conditions that depend on the values of the parameters. Choice of the algorithm itself adapts to the current performance of the allocated resources and the state of the performed tasks in the first phase. Thus, the proposed adaptive algorithm determines whether the domain decomposition or master-slave algorithm will be executed in the next phase based on the fulfillment of the defined conditions according to the principle: the more conditions are met, it determines the choice of DD algorithm in the third phase and vice versa.

In order to enable additional adaptation of the decision algorithm to a specific application and distributed system, each of the conditions can be weighted with real coefficients K_i, $K_i \in [0,1]$ which enables the exclusion of some conditions or assigning greater or lesser importance to some of the conditions. This does not apply to an eliminatory condition that is considered independently of the other conditions. The coefficients K_i are assigned a maximum value of 1 if this condition is fully taken into account, while $K_i = 0$ excludes the influence of this condition from the influence on the choice of algorithm. Coefficients should be defined separately for each application and distributed resource depending on previously obtained results and experiences.

Finally, based on the above conditions, we can define the decision function on the basis of which we select the algorithm in the third phase:

$$U = \sum_{i=1}^{4} Ki * Ui. \qquad (1)$$

The threshold value of the decision function U should also be defined, on the basis of which one or another algorithm is selected for the third phase (DD or MS). Since the maximum of the function U is achieved by the fulfillment of the conditions K_i*U_i and that determines the choice of the DD algorithm, then half of the maximum value of the function U is taken as the threshold value, ie

$$\mathrm{P} = \frac{\sum_{i=1}^{4} \mathrm{Ki}}{2}. \qquad (2)$$

Therefore, if it's satisfied

$$U \geq P \qquad (3)$$

it is necessary to select the DD algorithm in the third phase or the MS algorithm if condition is not satisfied.

Figure 2. shows a schema of the decision making process for the selection of algorithm in second phase. As already mentioned, based on the presented parameters, defined conditions and coefficients, the algorithm for the distribution of tasks in the third phase is selected.

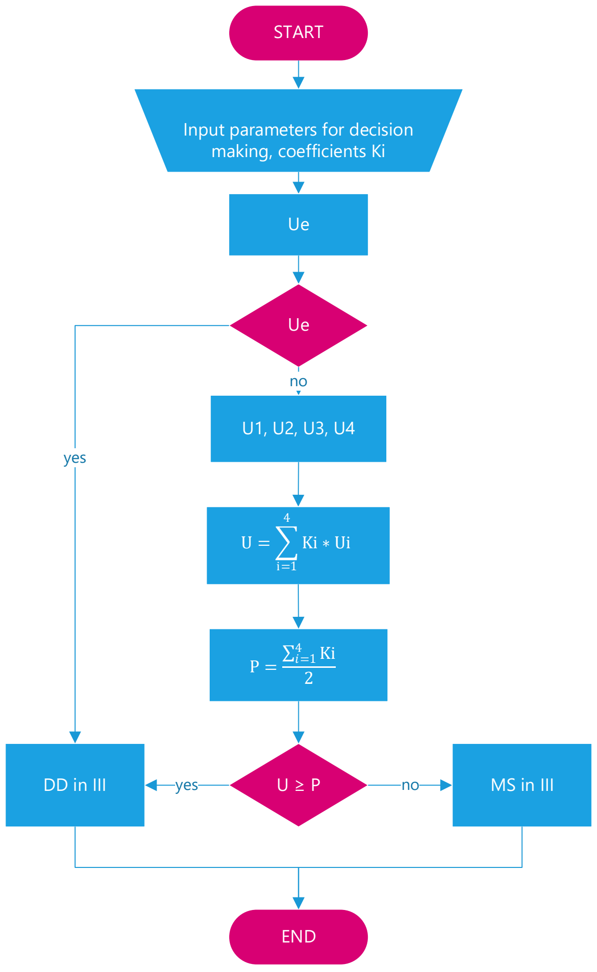
START
Input parameters for decision making, coefficients Ki
Ue
Ue
no
yes
U1, U2, U3, U4
$U = \sum_{i=1}^{4} Ki * Ui$
$P = \frac{\sum_{i=1}^{4} Ki}{2}$
DD in III
yes
U ≥ P
no
MS in III
END

Fig 2. Scheme of the decision making process for the selection of algorithm in Phase II

The selected algorithm (DD or MS) is executed in the third phase.

If the DD algorithm is selected, each core receives a portion of the list of unfinished tasks. Each core gets assigned one of the remaining tasks to solve if the remaining number of tasks is less than or equal to the available number of cores (condition Ue). Otherwise, the number of assigned tasks for each core is determined in proportion to the number of tasks completed in the first phase on each core separately.

In the case of selecting the MS algorithm, the core that performed the analysis in the second phase is determined as the master core. It contains information with a list of all unfinished tasks that are assigned to slave cores for execution in the third phase of the algorithm.

The proposed CAA algorithm will increase efficiency and shorten the execution time of parts of a parallel application in the third phase according to the interruption of the execution of the first phase, the analysis of the state of resources, the adaptation from the second phase and the redistribution of tasks.

The efficiency of the CAA algorithm has been improved due to process reallocation, reduced kernel latency for new instructions, and improved resource utilization by adapting the allocation to the distributed system architecture and application-specific. Therefore, the execution time of the proposed algorithm will be shorter than the execution time of the standard DD algorithm if measured under the same conditions. The CAA algorithm is similar to the CA algorithm in the case of deciding that a dynamic process allocation along with the MS algorithm is required in the third stage.

The disadvantages of the proposed CAA algorithm are the interruption of task execution at the end of the first phase and the duration of adaptation in the second phase. Interrupting the execution of tasks in the first phase may increase the duration of this phase if there are one or more tasks whose duration is significantly longer than the duration of other tasks. This phenomenon would cause an increase in the duration of the first phase, which may affect the performance of the entire algorithm. In that case, the efficiency would be the same as with the classical DD algorithm. The second phase, due to its short duration, cannot significantly affect the overall efficiency of the parallel application.

The proposed CAA works as a DD algorithm during the period of its maximum efficiency and stops working when its efficiency starts to decline. The proposed adaptive algorithm will have a significantly better performance than the domain decomposition algorithm in the case when the basic algorithm has low efficiency due to interruptions and redistribution of tasks.

The CAA algorithm will have better performance than the MS algorithm because the MS algorithm does not execute tasks on the master core and generates more communication losses than the proposed CAA algorithm. The MS algorithm will have lower efficiency than the proposed algorithm because it starts as a DD algorithm and redistributes and selects the algorithm for execution based on parameters in order to achieve better use of resources and efficiency.

In case of large losses during third phase, it is possible to re-initiate the interruption and repetition of the decision algorithm, ie adaptation based on new parameters, re-selection of the algorithm and its start to get the best use of resources.

IV. The analysis of simulation results

For the purposes of research and testing of the subject algorithms, a parallel version of the crossbar commutator performance simulator (CQ) [13] was used, as a numerically demanding example of a parallel application with several independent processes. The algorithms were tested on different distributed computing environments and run under different resource loads. Each simulation was performed ten or more times and the averaged results of the execution time are presented here. The performance of the combined adaptive algorithm was verified on the example of a 16-port CQ simulator with 1,000,000 requests and 3072 generated tasks. Simulations performed on the Paradox HPC cluster of the Institute of Physics in Belgrade. At the time of the simulation, the cluster consisted of 106 computing nodes based on two octa-core Xeon 2.6GHz processors with 32GB of RAM and NVIDIA® Tesla ™ M2090 cards. The performance of the combined adaptive algorithm is compared with the performance of the algorithms that make it up. Simulations were performed on 16, 32, 64 and 128 cores. The input files were copied to the nodes on which the simulations were run in the preprocessing phase, thus reducing the impact of communication between the nodes.

In the presented simulations, the value of standard deviation 10% of the average value of the core operating clock was used for condition U_1. A threshold of 32 cores is defined for condition U_2. For conditions U_3 and U_4, the value of the standard deviation is 25%. The coefficients used in these simulations are $K_1 = 0$, $K_2 = 1$, $K_3 = 1$ and $K_4 = 0.5$. Priority in decision making is given to the number of cores on which the simulation is performed and the number of performed tasks per core. A lower priority was given to the duration of the tasks, and due to the coefficient $K_1 = 0$, the influence of cluster homogeneity was not taken into account. The average results of parallel application execution with DD, MS and CAA algorithm for different number of used cores are shown in Figure 3.

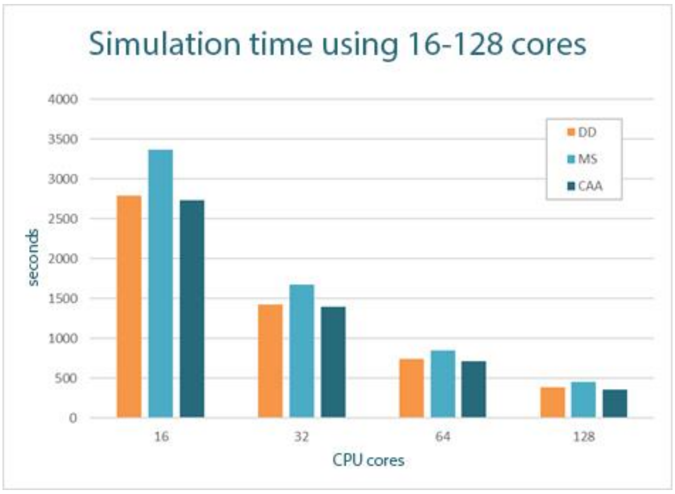

Figure 3. Average execution time of simulations using DD, MS and CAA algorithms on 16-128 cores

The combined adaptive algorithm completed simulations faster than the domain decomposition and master-slave algorithms in all conditions. The best results and the greatest benefits due to the redistribution of tasks were determined in cases of performing simulations on a number of cores. The simulations showed the longest execution time with the master-slave algorithm, especially on a small number of cores due to its previously described shortcomings.

The domain decomposition algorithm performed simulations faster than the master-slave algorithm. The input data was transferred before the simulations and most tasks were performed at approximately the same time, as shown in Figure 3. Therefore, the static distribution proved to be sufficient and the domain decomposition algorithm showed better performance than the master-slave algorithm.

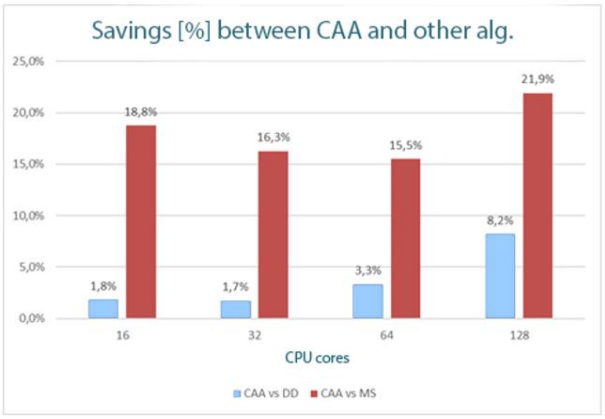

Figure 4. Savings during algorithm execution and comparison between combined algorithm and domain decomposition and master slave

Figure 4 shows the execution time savings between the combined adaptive algorithm and the algorithms that make it up. The domain decomposition algorithm required more time than the combined adaptive algorithm due to the static distribution throughout the execution process. The difference between the combined adaptive and domain decomposition algorithms ranges from 1.7% to 8.2%. The biggest difference was recorded when executing the application on 128 cores.

The differences between the combined adaptive algorithm and the master-slave algorithms are due to the loss of the master-slave algorithm due to the distribution of tasks and communication between cores during the entire program execution process. The execution time difference between the combined adaptive and master-slave algorithms ranges from 15.5% to 21.9%. The inability to execute tasks on the master core produced losses during execution on a smaller number of cores. Increased communication between cores throughout the execution of the simulation caused the largest difference between the results listed on 128 cores.

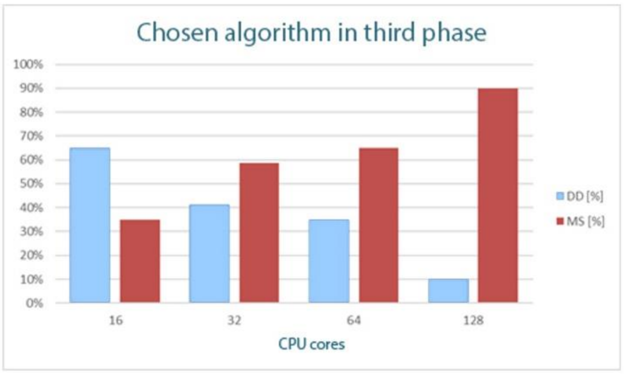

Figure 5. Selected algorithm in the third phase of CAA

Figure 5. shows the results of the selection of the algorithm in the second phase according to the received and analyzed data and the decisions made at the end of the second phase. The domain decomposition algorithm was chosen in most cases when the simulation was performed on 16 cores, because the execution was detected on less than 32 cores and an even number of tasks that needed to be redistributed. On the other hand, master-slave was chosen in cases of simulations on 32 or more cores because the decision algorithm from the second phase based on parameters discovered the number of available cores, different number and duration of performed tasks and selected this dynamic algorithm for the third phase.

V.Conclusion

The paper presents an original adaptive load balancing algorithm for parallel applications that combines the operation of static and dynamic algorithms. Domain decomposition and master slave algorithms were used on the basis for the proposed algorithm, as one of the most common algorithms in practice. As none of the algorithms provides good results in a wide range of applications and types of distributed systems, the following research was based on the idea of combining the mentioned algorithms in order to improve the parallelization performance without complication of the algorithm. Based on the identified advantages and disadvantages of standard algorithms, a combined adaptive algorithm is proposed. The idea of combined algorithms is to work in the phases when composite algorithms have the best performance. The advantages of the proposed solution are following:

- improved parallel application efficiency and cluster utilization in relation to basic algorithms due to task redistribution and reduced execution time;
- parameters and conditions for the selection of algorithms have been identified according to the status of resources and the point of execution of the application and determine a more adequate static or dynamic distribution of the process by an adaptive strategy
- weighting coefficients (K_i) adjust the adaptive load balancing algorithm and parallel application to the infrastructure
- applicability of the proposed adaptive part of the decision algorithm is possible in any load balancing algorithm and
- the proposed algorithm is applicable to all parallel applications consisting of several independent tasks.

The paper presents the results of executing domain decomposition, master-slave, combined and combined adaptive algorithm on different computer resources with the help of numerically demanding parallel application of CQ simulator. Comparison of the results of simulations with different loads and configurations of distributed resources confirms the better performance of the proposed algorithm in relation to the basic algorithms considered in the paper.

Acknowledgment

This research is supported in part by the EuroCC project, grant agreement grant agreement 951732 EuroCC-H2020-JTI-EuroHPC-2019-2.

References

[1] S. Tanenbaum and M. van Steen, Distributed Systems: Principles and Paradigms, 2nd Edition, Pearson Education. Inc., 2007.
[2] B. Barney, Introduction to Parallel Computing, Lawrence Livermore National, 2012.
[3] L. Filipovic, “Combined adaptive load balancing algorithm for parallelization of applications”, PhD thesis, University of Montenegro, Faculty of Electrical engineering, 2019.
[4] L. Filipovic and B. Krstajic, "Combined load balancing algorithm in distributed computing environment," Information Technology and Control, vol. 45, no. 3, pp. 261-266, 2016.
[5] H. D. Karatza and R. C. Hilzer, "Parallel Job Scheduling in Homogeneous Distributed Systems," Simulation, vol. 79, 2003.
[6] J. Dinan, S. Olivier, G. Sabin, J. Prins, P. Sadayappan and C.-W. Tseng, "Dynamic Load Balancing of Unbalanced Computations Using Message Passing," in Parallel and Distributed Processing Symposium, 2007, IPDPS 2007, IEEE International, Long Beach, CA, USA, 2007.
[7] T. Rauber and G. Rünger, Parallel Programming for Multicore and Cluster Systems, Springer, 2010.
[8] D. Thiébaut, Parallel Programming in C for the Transputer, 1995.
[9] V. Sarkar. Partitioning and Scheduling Parallel Programs for Multiprocessors. MIT Press, 1989.
[10] W. D. Gropp. Parallel Computing and Domain Decom-position. In: Fifth Conference on Domain Decompo-sition Methods for Partial Differential Equations, 1990, pp. 249-361.
[11] S. Sahni. Scheduling Master-Slave Multiprocessor Systems. IEEE Transactions on Computers, 1996, Vol. 45, No. 10, 1195-1199.
[12] L. Filipovic, B. Krstajic. Modified master-slave algorithm for load balancing in parallel applications. ETF Journal of Electrical Engineering, 2014, Vol. 20, No. 1, 74-83.
[13] M. Radonjic and I. Radusinovic, "CQ Switch Performance Analysis from the Point of Buffer Size and Scheduling Algorithms," in Proc. of 20th Telecommunication Forum TELFOR 2012, 2012