ECCOMAS Congress 2016
VII European Congress on Computational Methods in Applied Sciences and Engineering
M. Papadrakakis, V. Papadopoulos, G. Stefanou, V. Plevris (eds.)
Crete Island, Greece, 5–10 June 2016

ON A DYNAMIC SCHEDULING ALGORITHM FOR MASSIVELY PARALLEL COMPUTATIONS OF ATOMIC ISOTOPES

Elizaveta Dorofeeva[1], **Jatin Arora**[2], **Stefan Typel**[3], **Gevorg Poghosyan**[1], **Peter Sanders**[4], **Achim Streit**[1]

[1] Steinbuch Centre for Computing, Karlsruhe Institute of Technology
Hermann-von-Helmholtz-Platz 1, 76344, Eggenstein-Leopoldshafen, Germany
e-mail: elizaveta.dorofeeva@kit.edu

[2] PEC University of Technology
Sector -12, Chandigarh, 160012, India

[3] GSI Helmholtz Center for Heavy Ion Research
Planckstraße 1, 64291, Darmstadt, Germany

[4] Institute of Theoretical Informatics, Karlsruhe Institute of Technology
Kaiserstraße 12, 76131, Karlsruhe, Germany

Keywords: High-performance computing, Scalability, Scheduling, Nuclear clusters.

Abstract. *The goal of the work is to enhance the execution scalability of the code called AtomicClusters for evaluations of in-medium properties of nuclear clusters to extreme scales. In order to fully exploit the computing power of massively parallel supercomputing systems, the code was supplemented with parallel output and dynamic scheduling system based on task-stealing technique. The scheduling system was implemented for state-of-the-art distributed-memory high-performance computers (HPC) using the advanced features of Message Passing Interface (MPI) as an independent adjustable module. The parallel output was integrated into the code using MPI IO. A number of strong scaling tests was performed for the resulting parallel software. An almost linear scalability was reached on up to 4000 cores. The code scales up to at least 38400 processes, but with lower speedup. The obtained results are discussed in the fifth section of the paper.*

1 INTRODUCTION

The dynamical evolution of core-collapse supernovae, the static properties of neutron stars, and the conditions for nucleosynthesis could be determined by Nuclear Equation of States (EoS) of stellar matter [1], the chemical composition of stellar matter. At densities below nuclear saturation density the nuclear matter is a mixture of nucleons, nuclei and electrons that could be considered as huge atoms also known as nuclei clusters. Their abundances depend strongly on the thermodynamic conditions such as density, temperature, and electron fraction. Since nuclei, which are composite objects, are immersed in a dense medium, their properties change significantly with the parameters mentioned due to the nuclear and electro-magnetic interaction with the surrounding particles and the action of the Pauli Exclusion Principle [1]. The latter is the main cause for the dissolution of nuclei with increasing density (Mott effect) [1]. Hitherto, several thousand clusters have to be considered for various values of the thermodynamic parameters as well as atomic mass and charge number to identify stable ones.

The whole range of parameters for the evaluation of in-medium properties of all possible nuclear clusters is almost 500 million variations. Thus, the massive parallel calculations are crucial to complete full analyses. To reach good load balance of such computations at extreme scales, a middleware between cluster scheduling system and the simulation software, based on scalable scheduling algorithm has to be developed. Since the scalability of the commonly used single-queue master-managed technique is limited due to the Amdahl's law [2] (the work distribution can be done only sequentially as there is only one master), and the calculation time for each set of parameters of computation for nuclear clusters cannot be accurately predicted (see chapter 3.1), the usage of static scheduling algorithms proved to be inefficient and a principally different scheduling approach known as task stealing was used.

The task stealing suspects each process having own task stack, accessible concurrently by all the processes in system. Shared-memory systems meet the requirements of such an approach, but do not give enough computational resources for the simulations (the Blacklight machine of the Pittsburgh Supercomputing Center (PSC), the world's largest shared-memory computing system [3], features 512 eight-core Intel Xeon 7500 processors). Distributed-memory systems offer enough performance, but do not support direct access to the operative memory of another process. Therefore, a hybrid concept of the shared-memory-like synchronization based on MPI-3 passive-target remote memory access [4] was implemented.

The scalability tests of the designed parallel system were performed on the computational resource ForHLR Phase I (SCC/KIT, Karlsruhe) on up to 4000 cores and on the Cray XC40 Hazel Hen system (HLRS Stuttgart, #8 in Top500) on up to 38400 cores, using renowned MPI libraries.

The paper is organized as follows. Section 2 shortly describes physical models used in the code. Section 3 presents overview of applicable scheduling techniques. Section 4 gives more technical details of the implementation of the chosen approach. Section 5 is devoted to the quality assessment of the re-designed software. Conclusions and future work prospects are given in the section 6.

2 CODE "ATOMICCLUSTERS" FOR THE EVALUATION OF IN-MEDIUM PROPERTIES OF NUCLEAR CLUSTERS

A parallel *AtomicClusters* [5] code for the evaluation of in-medium properties of nuclear clusters was developed in a joint R&D project of the Steinbuch Centre for Computing of KIT with the Nuclear Theory group of GSI and is used to model the behavior of composite particles at zero and different non zero temperatures. The code allows searching for stable isotopes

and defining EoS consisting of such isotopes at extremely dense matter. This kind of EoS is an essential ingredient in astrophysical model simulations.

The main objective of the *AtomicClusters* code is to calculate the change of properties of nuclei, mainly energy and radii, embedded in a medium of nucleons and electrons, required to have charge neutrality of the total system as in stellar matter, for given temperature and (baryon) density. This is achieved by comparing the results of two calculations: one of homogeneous nuclear matter and one where the formation of a cluster is calculated inside a spherical Wigner-Seitz cell (inhomogeneous matter). For this we use a relativistic energy density functional (EDF) that is based on a phenomenological relativistic mean-field (RMF) model [6] for nuclei with density-dependent nucleon-meson couplings. The exchange of mesons models the interaction between the nucleons. From the EDF the relevant field equations, i.e. coupled differential equations, for the particles are derived. They have to be solved self-consistently. This is done using the relativistic Thomas-Fermi approximation. The results for energy shifts etc. are needed for the complete chart of nuclei, including unstable exotic nuclei, for different densities and temperatures. They will be used then in a second step as input for a model for the equation of state [1] of stellar matter (generalized relativistic density functional) that incorporates the full table of nuclei.

So far, only simple approximations for the energy shifts are used, but they should be replaced in the future by more realistic results based on the underlying EDF.

The parallelization technique used in the initial version of the *AtomicClusters* code was based on static domain decomposition, and proved to be inefficient (see section 5).

3 SCHEDULING SYSTEM

The necessity of optimal and effective use of limited and expensive computational time on any HPC system is crucial. Efficient resource utilization can seldom be reached without long term re-engineering of the scientific simulation codes. The code rebuilt may result in changes of the scientific concepts and architecture, practically leading to fundamentally new software and possibly different scientific results. In order to advance the *AtomicClusters* code for state-of-the-art HPC systems and avoid affecting the underlying physical models, the external scheduling system was developed. As distinguished from code parallelization techniques, the scheduling system operates code-defined computational units instead of mathematical operations, thereby not influencing the simulation results.

For the *AtomicClusters* code the scheduler-managed unit (hereinafter task), is one set of possible isotope parameters, namely density, temperature, atomic number and electron fraction. Since the calculation time for each task is unknown, the simple statement of the scheduling problem in case of heterogeneous computational system can be represented as follows: the project consists of N independent tasks of variable size; the tasks must be distributed among M: M<<N equal resources so that the latest are used in a most efficient way: the makespan of the project is minimized and the resources utilization is maximized; the distribution method must scale up to M>>10000. Standard static algorithms as those solving the bin packing problem [7] cannot be used to schedule tasks of undefined size. To solve the problem, the following scheduling approaches were implemented and evaluated.

3.1 Static scheduling techniques

The first approach is a static distribution of equal N/M number of tasks among processes, or static domain decomposition. The advantages of the approach are the simplicity and the lack of bottlenecks. Since the input range of parameters is represented in an incremental form, it is small enough to fit into the memory of a single process irrespective of the problem size,

and the distribution of parameters among processes is done in parallel, namely the range of parameters to be processed is calculated on each process according to its rank. The disadvantages of the approach are the non-optimized completion time and the high load imbalance on the computational systems with static resource management. This technique gives better result on systems with dynamic resource management, where the finished processes can be freed immediately without waiting for the whole computation to complete [8]. Herewith, fulfilling the resource occupation condition we are still not satisfying the makespan minimization criteria.

The second approach is a statistics-based grouping technique. The sizes of new tasks are estimated using the 3D spline-polynomial interpolation of the collected statistical data. The groups of tasks with similar computational intensity (cumulative size) are formed and scheduled as single units of equal size. Figure 1 demonstrates the estimated task size for the range of atomic numbers from 50 to 300 and temperatures from 25 to 140 MeV.

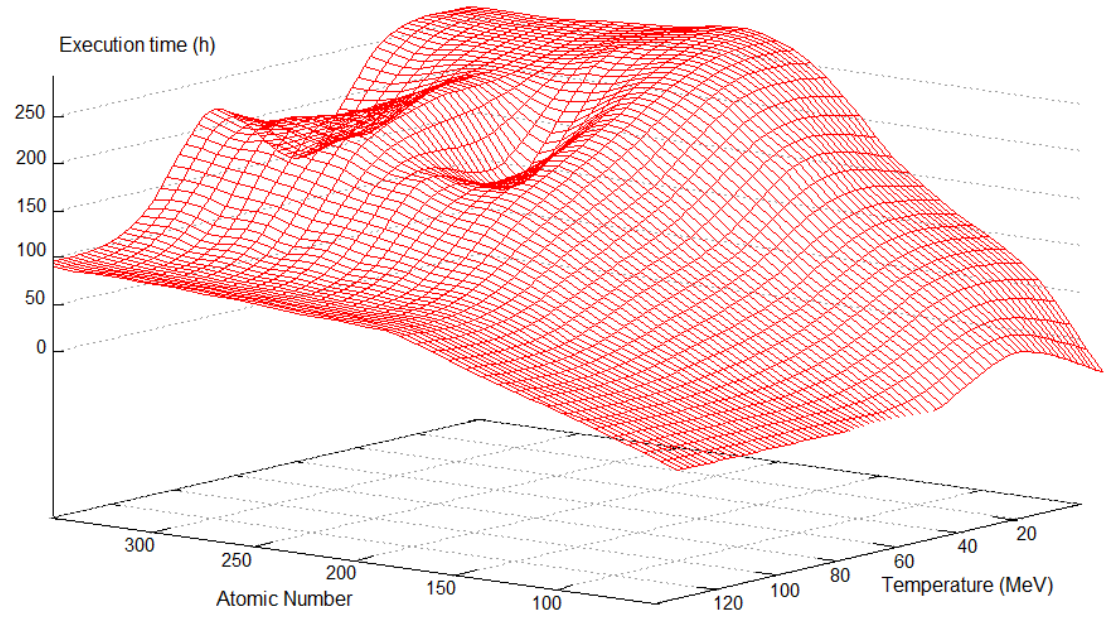

Figure 1: Estimation of task sizes for given temperature and atomic number.

On particular range of parameters, where the size function is smooth, the approach shows feasible results. However, the scalability of the approach is limited due to statistics I/O bottleneck, and the prediction is not possible on the whole range of parameters. This technique remains useful for the simulations where the task size dependency on the input parameters can be approximated with monotonous function, such as the software to search for periodic gravitational wave signals [9], where we successfully applied the grouping approach. In case of nuclear clusters study, where the computation of each set of parameters must be done only once, this approach must be advanced.

3.2 Dynamic task-stealing scheduling technique

Taking into account specifics of the *AtomicClusters* code such as non-predictable task sizes and independency of the tasks, the approach based on a task-stealing technique was chosen for the scheduling system.

The idea of the task-stealing technique is derived from work-stealing. In [10] work-stealing was opposed to work-sharing – the technique where a single scheduler is responsible for the work distribution. Both work-stealing and work-sharing do not meet the requirements we

have set. The work-sharing technique is master-managed and does not scale due to Amdahl's law. The work-stealing supposes that the tasks can be subdivided, that contradicts the idea of operating code-defined computational units.

The following scheduling algorithm was chosen to be implemented. The tasks are initially distributed among processes as in the first approach, but as soon as the process completes the tasks from own queue, it starts "stealing" tasks from other queues. The task-stealing algorithm is initially parallel since each process is following the same steps. Provided the communication time is negligible compared to computation time, the load imbalance is not higher than the size of the biggest task, since each process is occupied while there are tasks still left in the system. Contrary to the simple distribution, this approach conduces to the completion time minimization.

To balance the load, two techniques were applied: the random polling [11, 12], where the stealing requests are send to a randomly chosen target, and the ordered polling with match memorization, which means repetitive stealing from the same target until it is not empty. The first is quite efficient despite its simplicity and match unpredictability, but requires supplemental termination detection mechanism. The second is finite, since no new tasks are arising during the computational process, and the cycle can be subdivided into nested loops, first searching within the immediately accessible area (e.g. shared memory region), thus profiting from the HPC architecture. However, ordered polling can cause work maldistribution, and therefore higher communicational load, since the number of non-empty targets will decrease faster.

4 IMPLEMENTATION OF THE TASK-STEALING ALGORITHM

In the era of HPC one of the biggest problem limiting parallel applications scalability is the synchronization problem. The first goal of the scheduler implementation was to minimize interprocessor communications, therethrough reaching higher scalability. The task stealing algorithm is naturally suitable for the shared-memory systems, since it suspects each process having its own task stack, accessible concurrently by all the processes in system. To apply the task-stealing technique for distributed memory systems and optimize the interconnection between processes, shared-memory-like synchronization based on MPI passive-target remote memory access (RMA) [4] was implemented.

The RMA is a new MPI feature, requiring an external progress engine to remain truly passive-target [13], and not fully supported by all MPI libraries. Normally the MPI progress engine on each process activates only when the process itself is doing an active MPI communication. Therefore, once initiated the passive-target access epoch stalls and can be processed only when the target calls an MPI routine. The external progress engine is responsible for the background progression, irrespective of the processes behavior, and allows avoiding deadlocks during passive-target access.

For the scheduling system to remain efficient irrespective of the HPC environment, the additional optionally activated synchronization mechanism was integrated. The mechanism periodically simulates MPI activity during the computational process, therethrough poking the MPI progress engine and activating stalled calls from other processes. Considering that no additional communication is done within the synchronization call, the increase of the single task computation time is negligible. The disadvantage of such a mechanism is a conflict of load balancing and synchronization. On the one hand, passive-target access epochs happening between the synchronization calls still stall, causing load imbalance, but on the other hand the meaningful decrease of synchronization periods may badly affect the computation time of each task. Therefore, the usage of the MPI libraries supporting asynchronous progress is preferable.

5 EVALUATION OF THE DYNAMIC SCHEDULING SYSTEM

The quality assessment of the re-designed parallel software was performed in a number of scalability tests. Figure 2 demonstrates the results of strong scaling of the code on the ForHLR Phase I system. The ordered polling with match memorization was chosen as a balancing technique. The tests were run with all three MPI libraries installed on the system: MVAPICH [13], Intel MPI and Open MPI. The default settings of the libraries were used. The speedup was measured relating to the sequential runtime with the same library and tuning parameters.

Among the libraries available on the system only MVAPICH fully supports the asynchronous progress. However, minimizing interconnection time to a negligible value by use of the external progress engine caused the decrease of performance. The cumulative runtime of all tasks on all processes increased. The Intel MPI library shows better results for the simulations with additional synchronization, which can be explained with optimal library configuration for the system.

The magenta line is given for comparison and refers to the scalability results of the original (based on static domain decomposition) parallel version of the *AtomicClusters* code without integrated scheduling system.

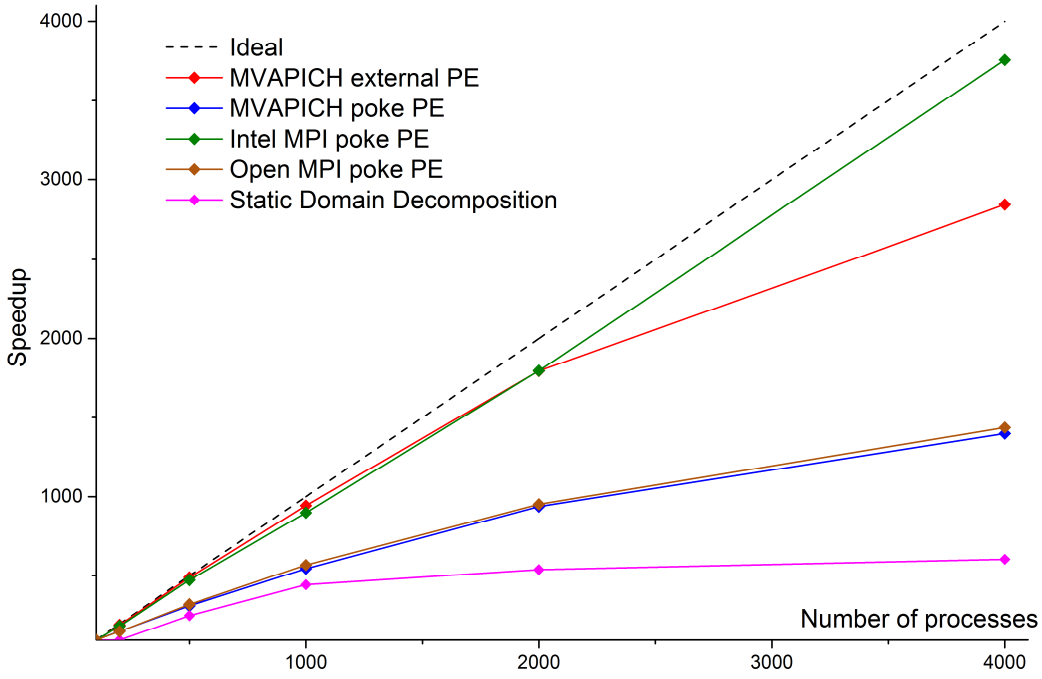

Figure 2: Scalability tests on ForHLR system with different MPI libraries.

The diversity of the results proves that to reach higher efficiency of the parallel application at any system, the initial tuning of the system software in order to optimize its settings for the used hardware and application must be done.

Figure 3 demonstrates the results of strong scaling of the code on the Cray XC40 Hazel Hen system with the default settings of the Cray environment.

Both load balancing mechanisms were tested on higher scales: the ordered stealing with repetitive target choice on match and the random polling. Additional synchronization was used to provide non-stalling passive target access.

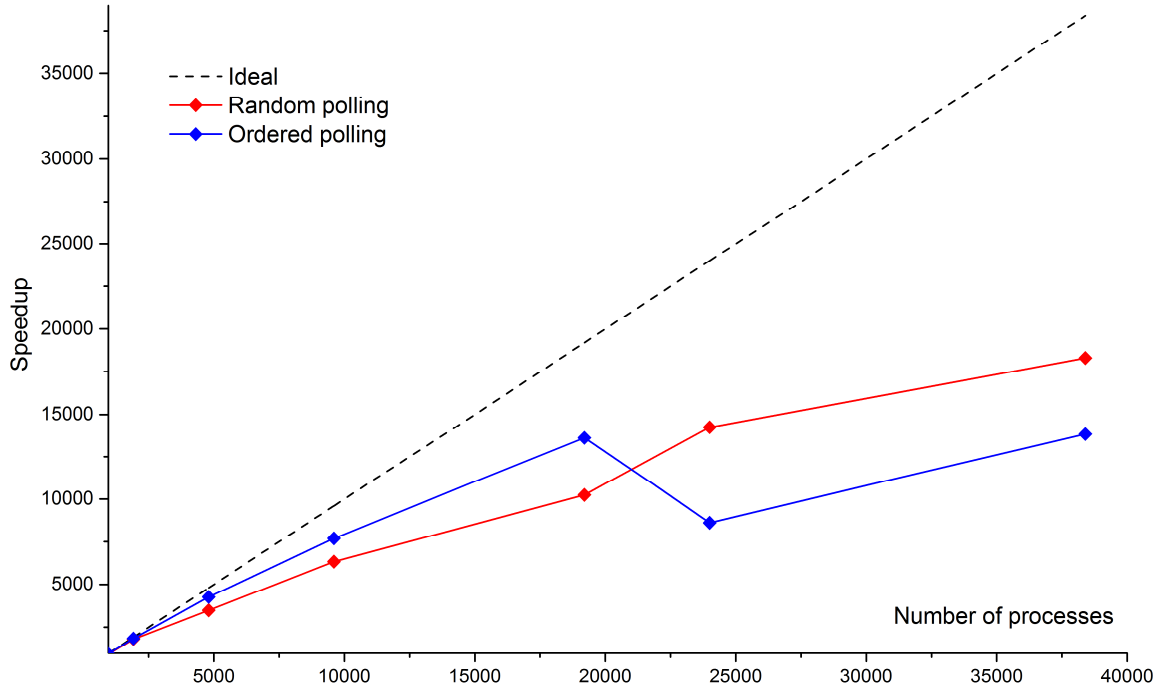

Figure 3: Scalability tests on Cray XC40 Hazel Hen system with different load balancing mechanisms.

As was predicted, the random target choice showed better results on scale-up. We explain it with the tasks maldistribution, caused by faster decrease of the number of non-empty targets, which is the consequence of repetitive target choice on match. This imbalance leads to higher number of access epochs to non-empty targets left and increases the number of concurrent access epochs. For random polling, the possibility of simultaneous epoch to the same target is quite low.

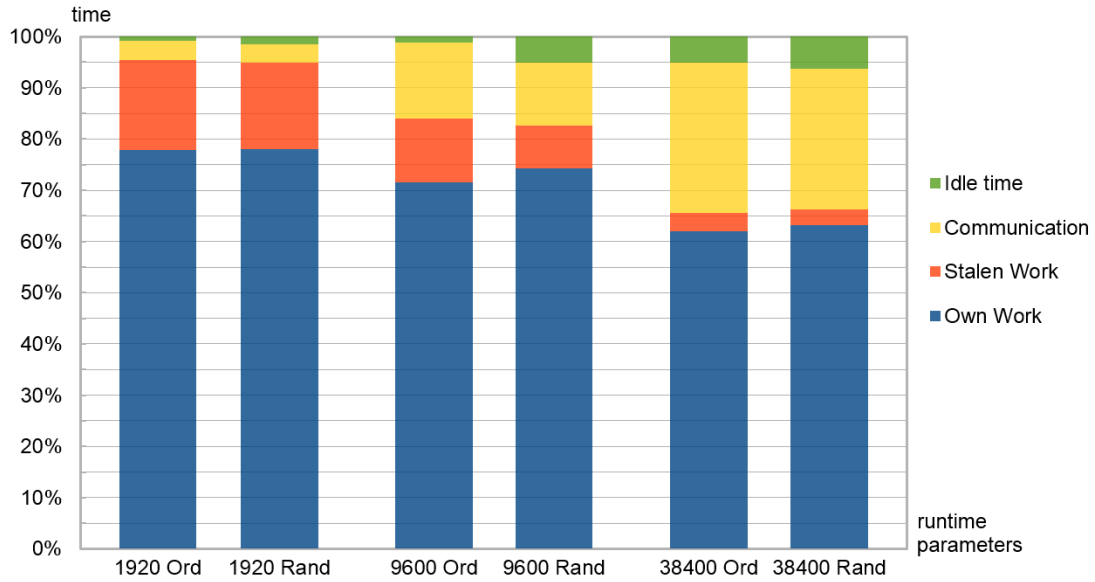

Figure 4: Relative work/idle/communication time distribution.

The overall result on Hazel Hen appeared to be worse than on ForHLR. Detailed evaluation of the results allowed discovering a bottleneck in the parallel output system, which can be

one of the reasons of speedup decrease. Here the optimization of the parallel output for *AtomicClusters* code is necessary and is under development. Another reason can be a not optimal (too small) size of the problem being solved. The performance of a single computational unit (core) of Hazel Hen system is almost two times greater than one of ForHLR according to the official documentation [14, 15], and is approximately three times greater according to the measurement of the single-process runtime of the same task on each machine. Thus the problem size must be enlarged to profit from Gustafson's law principles [16].

Higher performance also causes decrease of the computational time of a single task, which means that communication intensity increases. Figure 4 illustrates this increase, demonstrating the average distribution of processor time among work on own (received by initial distribution) tasks, work on stolen tasks, communication and staying idle, for different runtime parameters (number of processors and stealing order).

To minimize communications, the tasks grouping will be used so that groups rather than single tasks could be managed by scheduler as one unit. The optimal size of such a group as well as the preferable size of the whole simulated problem is a subject of further investigation.

6 CONCLUSIONS

The existing *AtomicClusters* code was supplemented with a dynamic scheduling system based on task stealing technique. The scheduling system was implemented using the advanced methods of MPI for the distributed memory system. The tests performed on the ForHLR system up to 4000 cores showed good scalability of the designed software. Results, obtained for higher amount of cores on the Hazel Hen system, allowed us appointing the following prospects of further system optimization:

- the revision of the parallel output system and its re-integration from code into scheduler;
- the estimation of the optimal size of the units operated by the scheduling system depending on the system performance;
- the investigation of the tuning parameters of the environments used and an evaluation of the optimal configuration.

Summarizing, one can claim that task stealing based scheduling concept appeared to be a good alternative to code parallelization and re-designing parallel software. Being used with optimal parameters it can provide almost linear speedup.

REFERENCES

[1] S. Typel, H.H. Wolter, G. Röpke, and D. Blaschke, Effects of the liquid-gas phase transition and cluster formation on the symmetry energy, *The European Physical Journal A*, **50:17**, 2014.

[2] G.M. Amdahl, Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities, *AFIPS Conference Proceedings*, **30**, 483–485, 1967.

[3] *Blacklight, the World's Largest Coherent Shared-Memory Computing System*, https://www.psc.edu/publicinfo/news/2010/101110_Blacklight.php, 2010, retrieved on 28-02-2016.

[4] W.D. Gropp, R. Thakur, Revealing the performance of MPI RMA implementations, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Springer Berlin Heidelberg, 272-280, 2007.

[5] G. Poghosyan, et al., High-Level Support Activities of Simulation Laboratory E&A Particles, *High Performance Computing in Science and Engineering*, **14**, 121-133, 2015.

[6] S. Typel, H.H. Wolter, Relativistic mean field calculations with density dependent meson nucleon coupling, *Nuclear Physics A*, **656**, 331-364, 1999.

[7] R. Lewis, A General-Purpose Hill-Climbing Method for Order Independent Minimum Grouping Problems: A Case Study in Graph Colouring and Bin Packing, *Computers and Operations Research*, **36 (7)**, 2295–2310, 2009

[8] W. Gropp, E. Lusk, and R. Thakur, *Using MPI-2 Advanced Features of the Message-Passing Interface*, The MIT Press, 1999.

[9] G. Poghosyan, S. Matta, A. Streit, M. Beiger, A. Krolak, Architecture, implementation and parallelization of the software to search for periodic gravitational wave signals. *Computer Physics Communications*, **188**, 167-176, 2014.

[10] R.D. Blumofe, C.E. Leiserson, Scheduling multithreaded computations by work stealing, *Journal of the ACM*, **46:5**, 720-748, 1999.

[11] P. Sanders, Randomized Receiver Initiated Load Balancing Algorithms for Tree Shaped Computations, *The Computer Journal*, **45:5**, 561-573, 2002.

[12] R. Finkel, U. Manber. DIB - A distributed implementation of back-tracking. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, **9(2)**, 235-256, 1987.

[13] G. Santhanaraman, S. Narravula, D.K. Panda, Designing passive synchronization for MPI-2 one-sided communication to maximize overlap. *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, 1-11, IEEE, 2008.

[14] *ForHLR Phase I - Hardware and Architecture*, https://www.bwhpc-c5.de/wiki/index.php/ForHLR_Phase_I_-_Hardware_and_Architecture, retrieved on 28-02-2016.

[15] Hazel Hen - Cray XC40, Xeon E5-2680v3 12C 2.5GHz, Aries interconnect, http://top500.org/system/178446, retrieved on 28-02-2016.

[16] J.L. Gustafson, Revaluating Amdahl's Law, *Communications of the ACM*, **31(5)**, 532-533, 1988.