Job Mapping Cyclic Composite Algorithm for Supercomputer Resource Manager

Anton Baranov[1,2](✉), Oleg Aladyshev[1,2], and Konstantin Bragin[1,2]

[1] Joint Supercomputer Center of the Russian Academy of Sciences, Branch of Federal State Institution Scientific Research Institute for System Analysis of the Russian Academy of Sciences, Moscow, Russia
{abaranov,o.aladyshev,kbragin}@jscc.ru

[2] National Research Centre Kurchatov Institute, Moscow, Russia

Abstract. One of the key functions of a supercomputer resource manager is to allocate jobs to the nodes within clusters or distributed computer systems. When running parallel jobs, different parts of the process interact with varying intensity, and how well the jobs are assigned to computing nodes directly impacts the efficiency of the job and the overall computing performance. Each time a job runs, the graph representing the application program needs to be matched with the graph of nodes that make up a subset of the computer system. Since both graphs are not known in advance, this mapping must be carried out within a reasonable timeframe while also managing resources. In our research, we have explored a cyclic composite mapping algorithm. This algorithm consists of two cyclic stages: parallel versions of simulated annealing and a genetic algorithm. Our experiments demonstrated that the proposed algorithm outperformed known mapping algorithms based on simulated annealing and genetic approaches in terms of both mapping quality and runtime.

Keywords: Supercomputer Job Management System · Resource Manager · Parallel Mapping Algorithm · Quadratic Assignment Problem · Simulated Annealing · Genetic Algorithm

1 Introduction

A modern supercomputer is comprised of interconnected computing nodes via a high-speed network. The network topology varies and includes communication channels with differing bandwidth and latency. This enables the representation of a supercomputer as a graph $G_s = (B, E_s)$, where B is a set of vertices representing computational nodes E_s is a set of edges representing communication channels between the nodes. The communication channel's throughput is described by the weights of the edges m_{ij}, where $i, j = 1, 2, ..., n$, n is number of edges, and $(i, j) \in E_s$.

https://doi.org/10.1007/978-3-031-78459-0_27

The primary function of a supercomputer is to run parallel application programs, which consist of processes that exchange information with varying degrees of intensity. A parallel program can be depicted as a graph $G_p = (A_p, E_p)$, where A_p represents a set of vertices corresponding to processes, and E_p represents a set of edges symbolizing the information links between processes. The number of processes is denoted as $N = |A_p|$. The edges of the graph are assigned weights $c_{ij}, i, j = 1, 2, ..., N$, reflecting the intensity of information exchange between processes i and j. This graph is referred to as a program graph or information graph. The representation of the mapping graph G_p of a parallel program onto the supercomputer structure graph G_s, is indicated as $\varphi : A_p \to B$. This mapping is expressed as a matrix $X = \{X_{ij} : i \in A_p, j \in B\}$, where $X_{ij} = 1$, if $\varphi(i) = j$, and $X_{ij} = 0$, otherwise. Assuming all computational nodes have equal performance, the optimal mapping criterion is the loss function $F(X)$ [1]:

$$F(X) = \sum_{i=1}^{N} \sum_{j=1}^{N} \sum_{p=1}^{N} \sum_{k=1}^{N} m_{ij} c_{kp} X_{ki} X_{pj} \to min \tag{1}$$

The supercomputer resource manager is responsible for allocating supercomputer resources to user jobs. It receives a stream of these jobs and places them in a queue. Each user job specifies the number of supercomputer nodes needed for running a parallel program. Before a job is launched, the manager allocates a subset of currently available nodes. Only after this allocation has been made, in accordance with (1), can the optimal mapping of the program graph onto the allocated nodes be determined. Therefore, the supercomputer resource manager is tasked with solving the mapping problem. Because it is not known in advance which nodes will be allocated for program execution, the manager must solve the mapping problem for each job that is launched.

In previous studies [1,2], a two-stage method for mapping a parallel program graph onto a computing system graph was considered. At the first stage, the manager selects the supercomputer nodes for the next job from among the available ones at the time of the job launch. At the second stage, the manager executes a parallel algorithm to map the program graph to the subsystem graph of the selected nodes on the subsystem before starting the user job. The study [2] considers such algorithm as a combined two-phase parallel algorithm. The first phase involves a simulated annealing algorithm, while the second phase utilizes a genetic algorithm.

The study [2] demonstrates that a parallel combined algorithm consistently provides a superior average solution compared to using a genetic algorithm and a simulated annealing algorithm independently. At the same time, the execution time of the combined algorithm is comparable to the execution time of the parallel genetic algorithm and significantly longer than the execution time of the simulated annealing algorithm. In this paper, to enhance the characteristics of the combined algorithm of [2], a cycle of phases of simulated annealing and genetic is proposed.

2 Related Works

The problem of mapping a program graph is widely recognized and is a special case of the Quadratic Assignment Problem (QAP) [3]. Numerous publications provide a variety of approaches and algorithms to solve QAP. The work [4] provides the classification of these algorithms. You can find detailed reviews of existing algorithms for solving QAP in [5,6]. The study [6] examines the suitability of algorithms for solving QAP for performance of supercomputers. The authors achieved a significant improvement in performance and energy consumption compared to the random assignment of parallel processes to supercomputer nodes. They also highlighted the potential benefits of combining different assignment algorithms.

Algorithms that find exact solutions, such as the full iteration method or the branch and bound method, always determine the minimum value of the loss function (1). However, as early as 1976 [7], it was shown that the assignment problem belongs to the NP problem class. The utilization of precise algorithms in supercomputer resource management is challenging because of the time required to identify a mapping graph, even for small or medium sizes.

For the supercomputer resource manager, approximation algorithms are relevant as they can find a solution close to the optimal value of the objective function in an acceptable time (1). Among the approximate methods, heuristic approaches are distinguished. The most common algorithm is selection (genetic algorithm), simulated annealing algorithm, and ant colony optimization algorithm. The previous work [2] provides a review of publications on this topic.

The tasks of building graphs of a computing system and a parallel program for the application of QAP solution algorithms are assigned to a separate direction. The methods for solving these problems are also discussed in reference [4]. The graph of a computing system is typically generated based on its architecture, which is described in the technical documentation. When the resource manager dynamically allocates computing nodes of a supercomputer, the required graph can be obtained by allocating a subgraph. A more reliable approach is to analyze the topology of a selected subset of nodes just before initiating a parallel program. This can be accomplished through test programs or data collected from the monitoring system. The graph of a parallel program in an executable configuration can also be obtained from the program's description or constructed using tools for analyzing the source code or the binary code of the executable file (referred to as «tracers»). [4] provides an analysis of the advantages and disadvantages of each method.

We would like to note the following recent publications. Nevertheless, before delving into that, it should be noted that in many works, what is referred to as a «hybrid algorithm» in this study is termed a «combined algorithm».

The work [8] proposes a heuristic algorithm for efficiently distributing MPI program processes across processor cores to minimize the total time for information exchanges. The study demonstrates a decrease in the execution time of MPI programs by employing the optimal mapping algorithm on computing clusters that utilize the Angara communication network [9].

In [10], the authors proposed a hybrid algorithm for solving QAP. The algorithm is based on the principles of genetic and evolutionary algorithms. The developed hybrid algorithm demonstrates superior accuracy compared to other heuristic approaches. The algorithm has demonstrated its effectiveness for the set of problem instances of small sizes $n = (12 - 25)$, with the solution accuracy deviating by no more than 4.2% from the known best solutions. Although the authors do not provide an analysis of the algorithm's execution time.

In reference [11], the authors implemented an evolutionary-genetic algorithm. Then, they proposed an improvement of the obtained solution by searching for local minima. As a result of the genetic algorithm and the corresponding initial solution, the best of the obtained solutions was selected. Then, a «path» was built between these two solutions, consisting of solutions obtained by paired permutations. In the event of a decrease in the value of the objective function, this decision was considered a local minimum, and permutations were made in it in order to find a better solution. The authors conducted an experiment using problem instances ranging from 25 to 100 in size. The combination of the genetic algorithm and further optimization of the solution proposed by the authors enabled them to achieve high mapping accuracy (average deviation is 1.63%) with a short solution time (when the problem size is 100, the time is 8.5 s). However, the operating time follows the laws of exponential growth, which raises doubts about the applicability of the algorithm to high-order graphs. The authors do not consider a parallel version of the algorithm.

In reference [12], the authors proposed a Routing-based Genetic Algorithm (RMGA). In order to find optimized solutions for the Task Mapping Problem, their work proposed an approach that considers a routing model as a fitness function instead of the classic (`flow_distance`) analytical evaluation to guide the search process. The inputs of the algorithm include application and computing system graphs, along with other parameters. The output of the algorithm is an optimized mapping solution. Initially, the algorithm creates a starting solution. Then, while the stopping criterion is not satisfied (cycle steps are less than the maximum objective function evaluations), the algorithm carries out crossover and mutation operations. They compared RMGA with other algorithms through experiments, demonstrating its effectiveness for almost all applications. They show gains ranging from 2% to 30% across different metrics.

Similar approaches were demonstrated in [13]. In this study, the authors employed a parallel genetic algorithm with certain modifications to address the assignment problem on a hybrid CPU–GPU system. They found that the GPU time is not significantly affected by increasing either the population size or the problem size compared to the CPU time. They tested their algorithm on the maximum problem size of 70. In addition, CPU time worsens when the population size is 600, becoming approximately twice the GPU time, and it increases significantly as the problem size grows. The authors did not utilize cross-node parallelism. It takes about 120 s to solve a problem with a size of 70.

The use of parallelism allows for improving the accuracy of the mapping simultaneously. In reference [14], the authors propose an algorithm based on the parallel Label Propagation Algorithm (LPA). The algorithm addresses the partitioning and mapping problems simultaneously. The authors considered large graphs with millions of vertices, significantly reduced them, and mapped graphs onto blocks of several hundred dimensions. In addition, this algorithm considers a specific mapping for hierarchical architectures, primarily for implicit tree topology.

In reference [15], the authors utilized a combination of a genetic algorithm and optimization of local solutions. The authors conducted an experiment that involved testing the technique on various sets of graphs. It is worth noting that even on average sets of a row of hundreds of vertices, the time to find a solution is hundreds of seconds, which is not feasible when planning the placement of the program on the nodes of a supercomputer.

The researchers in [16] note that the quality of the solution produced with machine learning techniques is not competitive compared to state-of-the-art metaheuristics. They also mention that there is still a long way to go before general learning techniques will surpass more direct optimization techniques for the QAP.

In the study [17], the authors utilize an improved hybrid algorithm that includes a genetic algorithm and an algorithm for simulated annealing, similar to how it is done in previous study [2]. They obtained a result indicating an improvement in the results of the combined algorithm compared to using pure annealing algorithms or genetic selection.

With a large number of papers devoted to the search for methods for the best mapping of a software graph to a graph of a computing system, researchers demonstrate an increased interest in solving the urgent problem of QAP. Heuristic methods are also applicable for supercomputers. At the same time, combined methods also show good results.

3 The Proposed Algorithm

The proposed algorithm is a further development of the works [1,4]. [4] offers the `GraphHunter` software tool designed to find the optimal mapping of a program graph to a multiprocessor system graph. `GraphHunter` is based on the free `UGR-Metaheuristics` [18] library. This library implements a variety of mapping algorithms. The `UGR-Metaheuristics` library is enhanced with classes that implement parallel simulated annealing algorithms, genetic algorithms, and combined algorithms. `GraphHunter` enables you to incorporate new mapping algorithms, select the most suitable parameters, and compare algorithms based on their accuracy and execution time. `GraphHunter` comprises several modules: loading program settings, constructing a supercomputer node graph distance matrix, building a program graph flow matrix, searching for a mapping, and generating a mapping file.

The combined algorithm added to `GraphHunter` is called PGSA (Parallel Genetic Simulated Annealing). The PGSA algorithm is executed in two phases: first, a parallel algorithm simulates annealing, and then selection is performed using a genetic algorithm. The simulated annealing phase is based on imitating the physical process that occurs during the annealing of metals. The essence of the algorithm is discussed in [2]. Following the algorithm, several processes (threads) search for solutions in parallel. This phase runs once in PGSA.

The second phase involves running a genetic algorithm. In the context of a genetic algorithm, an array p represents an individual. The i^{th} element (gene) of p contains the number of the node to which the i^{th} process of the parallel program will be assigned. Individuals constitute a population with a size equal to or greater than the number of vertices in the program graph. The crossover operation exchanges genes between two individuals in the population. The mutation operation changes a certain number of individuals with a specified probability. The selection operation chooses individuals from the population with the lowest value of the loss function (1). Parallel genetic algorithm runs for a given number of iterations, selecting the best solution in each process, and then chooses the best global solution.

The work [2] presents the results of an experimental comparison of the characteristics of three algorithms: simulated annealing, genetic selection, and combined PGSA. The accuracy of the display on graphs of small orders in the PGSA algorithm is comparable to the accuracy of the annealing algorithm. However, on graphs of large orders where the annealing algorithm does not perform well, the accuracy and speed of the PGSA algorithm are comparable to the genetic selection algorithm. At the same time, the PGSA algorithm was significantly inferior to simulated annealing in terms of search time. In order to enhance the characteristics of the PGSA algorithm, we have implemented several modifications to it. Given the results of [1] for graphs of small orders, we decided to loop the PGSA algorithm, repeating its phases several times. After finding a solution in the first iteration of the combined algorithm, the discovered mapping is reapplied to the input of this algorithm. Due to the random nature of the annealing algorithm, a new set of solutions is generated, which are then inputted into the genetic algorithm. The process of alternating between phases of simulated annealing and genetic selection is repeated several times. As a result, at each iteration of the new composite algorithm, the initial resolution demonstrates an increasing mapping accuracy from one iteration to the next. At every step, there is a solution that is getting closer and closer to optimal.

By analogy with the name in [2], we propose the name **Cycled PGSA** or **CPGSA** for the new algorithm. In short, CPGSA consists of the following phases:

1. Parallel search for solutions by each process using simulated annealing.
2. Generation of the initial population of solutions for a parallel genetic algorithm from step 1.
3. Running a parallel genetic algorithm for a specified number of iterations.
4. Selecting the optimal solution for each process.
5. Choosing the best global solution.
6. If it is not the last iteration, generate initial values for step 1.
7. Repeat steps 1–5 for the specified number of times.

The choice of CPGSA algorithm parameter values is based on recommendations from [4]. We have also introduced additional parameters. The most interesting parameter is the one responsible for the number of iterations. The execution time of the entire iterative CPGSA algorithm directly depends on the value of this parameter. Since the running time of the CPGSA must be comparable to the running time of the PGSA, in which only one iteration is performed, the number of repetitions must be selected accordingly to other parameters that significantly reduce the running time of each step.

4 Experimental Results

`GraphHunter` has been enhanced by incorporating the CPGSA algorithm into the mapping search module. Experiments were performed using the same `Broadwell` section of the MVS–10P [19] as in [2], following the methodology outlined in [2] and [4]. The `Broadwell` section comprises 136 nodes with the following features:

– 2 Intel Xeon E5-2697Av4 processors;
– 32 physical, 64 virtual cores in the node;
– 128 GB of RAM;
– Intel Omni-Path interconnect.

The same set of `Taixxeeyy` problem instances [20] was used as in [2] (`xx` represents the size or order of the instance, and `yy` represents the instance number). The computer system graph represents the distance matrix of an instance, while the program graph represents the flow matrix of that instance. For each pair of graphs utilized in our experiments, the minimum value of the loss function (1) is known.

In experiments, we utilized flow and distance matrices (graphs) from problem instances `tai27e01`, `tai45e01`, `tai75e01`, `tai125e01`, `tai175e01`, `tai343e01`, and `tai729e01` obtained from the website `Quadratic Assignment Instances`. http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/qap.html in the same configuration as in previous work [2] 256 cores of 8 `Broadwell` hosts. Figure 1 demonstrates the comparative results of the experiments. The best solution values for the `Optimum` curve in Fig. 1 can be found in the file on the website: http://mistic.heig-vd.ch/taillard/problemes.dir/qap.dir/summary_bvk.txt The values of the `Simulated annealing`, `Genetic`, and `PGSA` curves come from [2].

Figure 1 shows that on graphs of small and medium orders, the combined PGSA finds a solution comparable to the best solution obtained by the simulated annealing algorithm or the genetic algorithm. However, the proposed CPGSA offers a more precise solution for mapping a program graph to a multiprocessor system graph. The mapping accuracy, indicated by a smaller loss function, increases from 3% on graphs of small orders (`tai45` set) to 12% on graphs of medium orders (`tai125`). On the `tai175` instance set, the loss function values are comparable to the loss function value of the simulated annealing algorithm. At the same time, the running time of the CPGSA is much shorter (see Fig. 2).

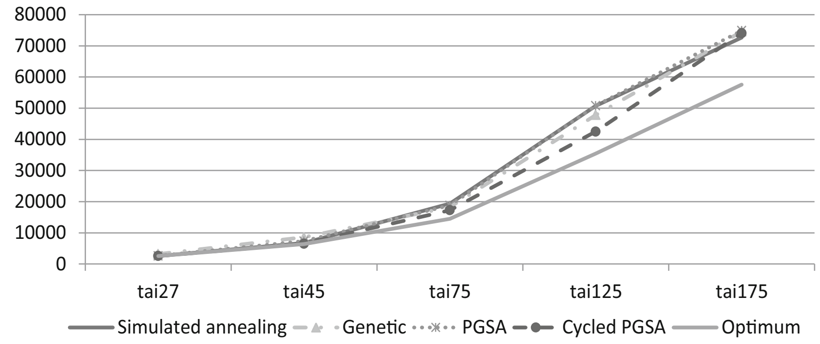

Fig. 1. The dependence of the loss function value on various program graphs and algorithms

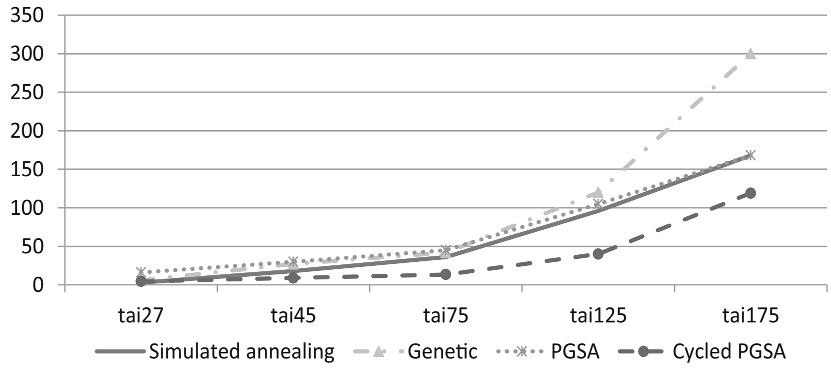

Fig. 2. The impact of program graphs and algorithms on running time

The graphs in Fig. 2 demonstrate that the cyclic composite algorithm can significantly reduce the time required to find a solution. Acceleration ranges from 1.41 times on a `tai175` set to 2.68 times on a `tai75` set. Such a decrease in the operating time of the algorithm significantly depends on the parameters utilized. In the experiments conducted, an approach was employed where multiple short iterations of the cyclic algorithm take place. Furthermore, optimizing the source code has enhanced the runtime.

5 Algorithm Parameters

Let us consider the nature of CPGSA using the `tai175` instance as an example. The solution accuracy increases iteratively, unlike the PGSA. Figure 3 shows the results of CPGSA.

The combined algorithm steps are repeating in a cycle. The solution obtained in the previous step is the input of the algorithm in the next step. Each iteration improves the solution slightly. The curve depicting the dependence of the loss function (1) for CPGSA on a step takes the form of a hyperbola, indicating that the solution is approaching the optimal one. The algorithm reaches a plateau after a large number of steps, finding an approximate solution that it cannot further improve.

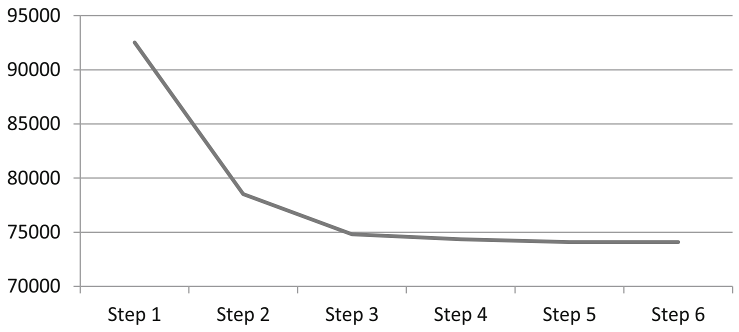

Fig. 3. The loss function for CPGSA depends on each step (iteration)

A small number of steps does not result in a significant improvement in mapping accuracy. The optimal number of steps (iterations) is 3–5. We have incorporated the step limit into the `GraphHunter` software accordingly. The solution found is considered the best when the accuracy does not improve in the next step, and the search is halted.

Let us consider the set of parameters used in the initial and following steps. It is possible to fine-tune the algorithm separately in the first step, which has a significant impact on the result, and in subsequent steps, which improve the solution obtained in the previous steps.

We propose changing the parameters of the simulated annealing and genetic phases at each step to solve the issue of the «stuck» cyclic composite algorithm.

Experiments have shown that changing the annealing temperature did not yield any results when CPGSA was applied to graphs of medium and large orders. Subsequent changes to other parameters from [2] were also ineffective. All parameters of the algorithm were changed simultaneously for this test. Modifiers were applied to all algorithms' parameters at each step. Modifiers are real numbers ranging from 0 to 2. For example, the initial population value in the genetic algorithm is 100. The modifier 0.8 will be applied to this value after the first step, and the value will be 80. Then, the population size will decrease at each step in accordance with the modifier and will be 64, 51, 41, 32, respectively. Thus, the CPGSA changes slightly at each step. This raises the task of finding modifier values.

We propose to automatically find values for modifiers using the simulated annealing algorithm. At each iteration, the value of the loss function of the CPGSA is compared with the value of the previous step. If the value decreased, the modifiers are preserved. If the value does not decrease, then we randomly change one of the modifiers and repeat the process. Over time, the available modifier change window narrows, allowing for more precise solutions to be obtained.

Note that we can run the process of searching for modifiers only once. Next time, when running on the same number of cores, we can use these modifiers to reduce running time. When running CPGSA on a different number of cores, new values of the modifiers need to be determined.

Figure 4 presents the results of modifier calculations for various instances. For instances `tai75` and `tai125`, we have found the optimal values of modifiers. For smaller instances, the search for modifier data is irrelevant because both the CPGSA and PGSA without modifiers find a solution that is close to optimal.

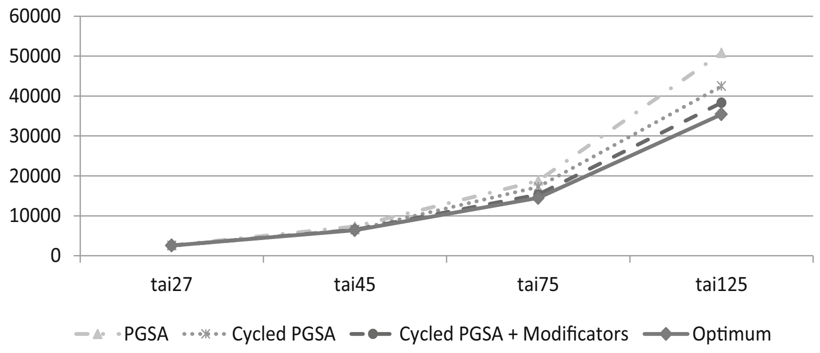

Fig. 4. The dependence of the loss function value on different algorithms across various instances

It is easy to see a significant improvement in accuracy compared to algorithms without modifiers. On the instance `tai75`, the accuracy of CPGSA increased by 19%, and the accuracy of PGSA increased by 22%. On the instance `tai125`, the accuracy increased by 10% compared to CPGSA and by 32% compared to the PGSA algorithm.

Figure 5 shows the solution search time in seconds for various algorithms and instances, where CPGSA is significantly faster than the PGSA and the CPGSA using modifiers. This is due to the nature of the search for a solution, the more time it takes to find a solution at different stages, the more accurate the resulting mapping can be.

It is worth noting that the deviation of accuracy from the optimal solution is 5% and 8% for the sets `tai75`, `tai125`, respectively.

6 Time to Map

The acceleration level of a parallel program, when optimizing its distribution across the nodes of a supercomputer, directly depends on the homogeneity of

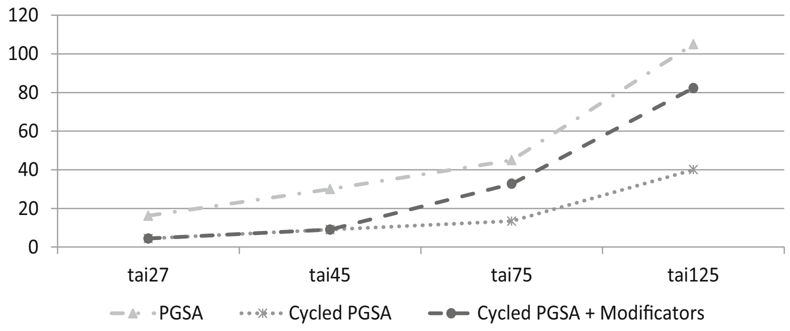

Fig. 5. Dependence of running algorithm time in seconds on various instances

the links between the nodes. Optimizing the placement of parallel program processes on supercomputer nodes can increase calculation performance by 10% to 25%, as demonstrated in [1,8]. According to statistics, the average task execution time on the supercomputer MVS-10P OP is 270 min. Thus, for MVS-10P OP, the placement optimization process should not exceed 10% of the program completion time, which is 27 min. This time aligns well with the system timeouts of the supercomputer resource manager, typically set at 10–15 min.

Since the convergence graph of the iterative CPGSA algorithm takes the form of a hyperbola, meaning it converges to a suboptimal solution with each new step, it is reasonable to interrupt its operation when the loss function value stops decreasing within a specified range or when the allocated time for the optimization process has elapsed. Based on statistical data, the mean time for MVS-10P OP is 27 min.

7 Conclusion

We utilized parallel algorithms that run on the nodes of the supercomputer assigned to the job to efficiently determine a mapping of a program graph to a supercomputer graph. We proposed a cyclic composite parallel (CPGSA) algorithm to search for a mapping. The CPGSA algorithm is based on a cyclic phase change of simulated annealing and genetic selection. At each iteration of CPGSA, the accuracy of the solution found increases. The proposed approach made it possible to achieve an improvement in accuracy and search speed. We propose using modifiers for the algorithm parameters at each step to enhance the performance of the CPGSA algorithm. A simulated annealing algorithm was used to find the optimal modifier values. As a result, it was possible to achieve higher accuracy in mapping the program's graph to the multiprocessor system's graph compared to using the CPGSA and the PGSA, all without altering the operating parameters. The ultimate goal of the study is to explore the optimal parameters of the CPGSA algorithm for the high-order QAP problem.

Acknowledgements. The research was carried out as part of government assignment of the National Research Centre «Kurchatov Institute».

References

1. Baranov, A.: Method and algorithms for a parallel program optimal mapping to the a multiprocessor computer structure. In: Proceeding of the High Performance Computing and its Applications Conference, Russia, Chernogolovka, pp. 65–67 (2000). (in Russian)
2. Baranov, A., Kiselev, E., Shabanov, B., Sorokin, A., Telegin, P.: Comparison of three job mapping algorithms for supercomputer resource managers. Lobachevskii J. Math. **43**, 2833–2845 (2022). https://doi.org/10.1134/S199508022213008X
3. Drezner, Z.: The quadratic assignment problem. In: Laporte, G., Nickel, S., Saldanha da Gama, F. (eds.) Location Science, pp. 345–363 (2015). https://doi.org/10.1007/978-3-319-13111-5_13
4. Baranov, A., Kiselev, E., Sorokin, A., Telegin, P.: A graphhunter software tool for mapping parallel programs to a supercomputer system structure. Softw. Syst. **35**(4), 583–597 (2022). https://doi.org/10.15827/0236-235X.140.583-597. (in Russian)
5. Leushkin, A., Neumark, A.: The quadratic assigment problem. Methods overview, generation test problem with known optimal solution. Proc. Nizhny Novgorod State Tech. Univ. n.a. R.E. Alekseev **4**(131), 26–34 (2020). https://doi.org/10.46960/1816-210X_2020_4_26. (in Russian)
6. Gupta, M., Bhargava, L., Indu, S.: Mapping techniques in multicore processors: current and future trends. J. Supercomput. **77**(8), 9308–9363 (2021). https://doi.org/10.1007/s11227-021-03650-6
7. Sahni, S., Gonzalez, T.: P-Complete approximation problems. J. ACM **23**(3), 555–565 (1976). https://doi.org/10.1145/321958.321975
8. Khalilov, M.R., Timofeev, A.V.: Optimization of MPI-process mapping for clusters with angara interconnect. Lobachevskii J. Math. **39**(9), 1188–1198 (2018). https://doi.org/10.1134/S1995080218090111
9. Mukosey, A., Semenov, A.: Simulation of utilization and energy saving of the angara interconnect. Lobachevskii J. Math. **43**, 873–881 (2022). https://doi.org/10.1134/S1995080222070186
10. Polupanova E., Nigodin E.: Hybrid algorithm for solving the quadratic assignment problem. Mod. Inf. Technol. IT-Educ. **17**(2), 315–323 (2021). https://doi.org/10.25559/SITITO.17.202102.315-323. (in Russian)
11. Bykova, M., Khloptsev, N.: Heuristic algorithm for solving the quadratic assignment problem. In: Proceedings of the XXVII International Scientific and Technical Conference Information Systems and Technologies – 2021, pp. 686–689 (2021). https://doi.org/10.46960/43912316_2021. (in Russian)
12. Rocha, H.M.G.D.A., Beck, A.C.S., Maia, S.M., Kreutz, M.E., Pereira, M.M.: A routing based genetic algorithm for task mapping on MPSoC. In: X Brazilian Symposium on Computing Systems Engineering (SBESC), pp. 1–8 (2020). https://doi.org/10.1109/SBESC51047.2020.9277843
13. Alfaifi, H., Daadaa, Y.: Parallel improved genetic algorithm for the quadratic assignment problem. Int. J. Adv. Comput. Sci. Appl. (IJACSA) **13**(5) (2022). https://doi.org/10.14569/IJACSA.2022.0130568

14. Predari, M., Tzovas, C., Schulz, C., Meyerhenke, H.: An MPI-based algorithm for mapping complex networks onto hierarchical architectures. In: Sousa, L., Roma, N., Tomás, P. (eds.) Euro-Par 2021. LNCS, vol. 12820, pp. 167–182. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-85665-6_11
15. Miseviius, A., Veren, D.: A hybrid genetic-hierarchical algorithm for the quadratic assignment problem. Entropy **23**(1), 108 (2021). https://doi.org/10.3390/e23010108
16. Luong, T., Taillard, É.: Unsupervised machine learning for the quadratic assignment problem. Lect. Notes Comput. Sci. **13838**, 118–132 (2023). https://doi.org/10.1007/978-3-031-26504-4_9
17. Türkkahraman, Ş., Öz, D.: An improved hybrid genetic algorithm for the quadratic assignment problem. In: 6th International Conference on Computer Science and Engineering (UBMK), pp. 86–91 (2021). https://doi.org/10.1109/UBMK52708.2021.9558978
18. Metaheuristica — Practica 3.a. https://raw.githubusercontent.com/salvacorts/UGR-Metaheuristics/P3/doc/memoria/memoria.pdf. Accessed 06 June 2024
19. Savin, G.I., Shabanov, B.M., Telegin, P.N., Baranov, A.V.: Joint supercomputer center of the Russian academy of sciences: present and future. Lobachevskii J. Math. **40**(11), 1853–1862 (2019). https://doi.org/10.1134/S1995080219110271
20. Drezner, Z., Hahn, P., Taillard, É.: Recent advances for the quadratic assignment problem with special emphasis on instances that are difficult for meta-heuristic methods. Ann. Oper. Res. **139**(1), 65–94 (2005). https://doi.org/10.1007/s10479-005-3444-z