ADVANCES IN CYBER-PHYSICAL SYSTEMS

Vol. 3, No. 1, 2018

ANALYSIS AND OPTIMIZATION OF THE SIZES OF THE ITERATION SPACE TILES DURING THE PARALLELIZATION OF PROGRAM LOOP OPERATORS

Alexander Chemeris, Sergii Sushko

Pukhov Institute for Modeling in Energy Engineering, Ukraine
Author's e-mail*: a.a.chemeris@gmail.com*

Submitted on 09.12.2018

***Abstract*: The analysis of the dependency of influence of the tile sizes of iteration space has been represented. It involves program loop operators' modification during parallelization for multithreading architectures of the computation systems. The particle swarm optimization method has been considered as a method of the minimization program execution time for tiling speed up.**

***Index Terms*: software optimization, loop parallelization, tiling, particle swarm optimization.**

INTRODUCTION

Software optimization is a process of modifying or transforming of source code to achieve a specific goal. It can be treated as a reduction of the program execution time, improving performance, reducing program code, saving memory, minimizing energy costs and reducing the input/output operations.

Software optimization is a multistage process. The main stages of optimization can be identified as:

- algorithm-level optimization,
- command-sequence optimization,

optimization of the processor operations.

The implementation of the program optimization process is performed by using a sequence of the different optimizing transformations, algorithms that take a program and modify it to obtain a semantically equivalent version of it but more effective in terms of any set of the optimization goals. In [1] it was shown that some code optimization problems are NP-complete or even insolvable. However, in practice, many of them are solved by heuristic methods, which give a result in a satisfactory time of processing of the source code of the program.

Optimization at the algorithm level is an improvement on the overall chain of the computations. For example, fast Fourier transform is an improvement of the discrete Fourier transform. Optimization of this level is usually performed by software developer or mathematician. Optimization at the command sequence has a great interest since sequence of operations can vary widely and the result will be unchanged. Optimization on the level of the processor operations is a process of representing of high-level language operators with some executable processor code. This job is usually performed by the compilers.

Optimization tasks on the command level are studied and solved almost from the moment of the compilers have been invented. Each compiler in fact does not just convert high-level language commands into machine codes or other executable codes, but also makes changes in the sequence of executable code. It can make either rather simple changes, such as unrolling a small loop into a sequence of unconditional operations or complex operations – converting several arithmetic or logical operations into others, converting a sequence of computations, splitting computing tasks into parallel kernels, and many others. Efficiency of compiler depends largely on the optimizations at this stage.

In addition to optimizing programs at compile time and creating executable code, other approaches to improving program parameters have been used. The first one is an automatic parallelization of the sequential algorithms and creation during the process of the parallel program for the specific computational architecture. Another approach is a process almost independent of the architecture of the computing system. In this case, an output of the optimization process is a generation of some transformed program of the high-level language. This program can be optimized both sequential and parallelized for execution on the multiprocessor computers. Since the main computational work has been concentrated in loops, parallelization and optimization of loops is the most perspective task.

SOFTWARE OPTIMIZATION PROBLEM

The optimization problem for each *i*-th computing loop can be determined by the formula:

$$F_i = argmin(f(\vec{M}_i, \vec{P}_i)) \quad (1)$$

where F_i are parameters that should be optimized, M is vector of the optimization methods, P is vector of the parameters of the optimization vectors. Function *argmin* is a set of points x, for which $f(x)$ attains the function's lowest value (if it exists):

$$\underset{x}{argmin}(f(x)) \in \{x \mid \forall y: f(x) \leq f(y)\} \quad (2)$$

Optimization problem for the entire program which consists of *N* computational loop in this case is determined by the formula:

$$F = argmin(\sum_{i=1}^{N} f(\overrightarrow{M_i}, \overrightarrow{P_i})). \quad (3)$$

III. ITERATION SPACE

Most of the computations in programs are concentrated in the loops that deal with index variables representing *n*-dimensional data arrays – vectors, matrices, etc. Loop parallelization is the most essential resource for the increasing computing performance.

In general, we consider nested loops, each of which defines its own index variable. The set of index variables defines an index vector of the loop nest $I = (I_1, I_2, .., I_n)$. For any loop, where index of loop *I* varies from value of *L* to *U* with step *S*, an iteration number *i* equals to the value *(I - L + S) / S*, where *I* is an index value for this iteration.

For *n* nested loops iteration vector *I* for the innermost loop is a vector containing an integer number of the iterations for each loop in order for the loops to be nested. In other words, an iteration number of multidimensional nested loop is determined according to the form $I = \{i_1, i_2 \ldots., i_n)$, where i_k is the iteration number of the loop for the nested level *k*.

Definition 1. The iteration space of loop is a set of all integer vectors $I = (I_1, I_2, \ldots, I_n)$ that satisfy the system of inequalities

$$Li \leq xi \leq Ui, \quad (4)$$

where $i = 1 \ldots n$.

Inequalities (4) define the boundaries of loops and limits of iteration space by convex polyhedron.

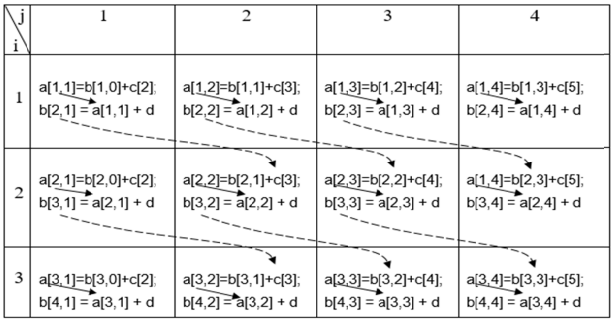
Definition 2. Let two occurrences of the variables u and v have d common loops. Consider set of all iterations $\boldsymbol{I_k} = (I_{k1}, I_{k2}, \ldots, I_{kn1})$ and $\boldsymbol{J_k} = (J_{k1}, J_{k2}, \ldots, J_{kn2}$, where $v[\boldsymbol{J_k}]$ depends on $u[\boldsymbol{I_k}]$. Each vector $(J_{k1} - I_{k1}, J_{k2} - I_{k2}, \ldots, J_{kd} - I_{kd})$ (for particular k) let it be called a distance vector of v of u and a set of the different vectors $D = (J_{k1} - I_{k1}, J_{k2} - I_{k2}, \ldots, J_{kd} - I_{kd})$ will be called as set of distance vectors for the dependence of v on u.

Distance of dependency plays an important role in the analyzing loops for parallelism. This value makes it possible to determine type of the data dependence and possibility of splitting of the iterative space into responsibility zones for the parallel execution. The body of loop consists of set of operators S. Dependencies between operators exist both inside loop and between iterations. Let's describe unfolded few steps of execution of the loop operator in the iteration space.

In Fig. 1a an example of loop is shown and Fig. 1b has its iterations, where dependencies inside a loop body are represented by the solid lines and dotted lines represent the inter-iteration dependencies. For the array a there are data dependencies. While for the array b there are inter-iteration data dependencies.

```
for( i=1; i<N; i++)
    for(j=1; j<N; j++) {
        a[i,j] = b[i,j-1] + c[j+1];
        b[i+1,j] = a[i,j] + d;
}
```

a



b

Fig. 1. Dependencies inside loop and between iterations

In the nest of *n* nested loops a set of distance vectors is approximated by an integer value *l* from the interval $[1, n] \cup \{\infty\}$, which is defined as the largest integer, so that the first $l-1$ components of the distance vectors are zeros. Dependence at the level $l \leq n$ means that dependence is found at the level *l* of the loop nest, that is, at a given iteration $l-1$ of external loops. In this case, it is said that the dependence is an inter-iteration dependence and such dependencies are called loop-carried at level *l*. If $l = \infty$, then dependence occurs inside the body of loop, between two different operators. Such dependencies are called loop-independent. The value of *l* is called as level of dependency.

Thus, define iteration space as a directed graph in which vertices are operators of the loop body and arcs are dependencies between these operators. Then the paralleling problem is represented in the mathematical formulation as the problem of cutting a graph into sub-graphs. In order to do this, well-known mathematical optimization methods can be used. Cutting a graph into sub-graphs, the methods for its transformation and parallelization can be considered.

One of the effective models for working with computational loops is a polyhedral model. Polyhedral model allows to apply various optimization methods to the computational loops. At the same time, the model itself operates with the concepts of set theory, affine transformations and Presburger arithmetic [2]. Presburger arithmetic is the theory of describing integers including {=, <, +, 0, 1} and it operates with such concepts as a tuple, set, relation. Based on this approach, an iterative loop space can be built and processing operations of such space are provided.

As an example, consider a loop whose code is shown in Fig. 2a. Fig. 2b represents the relationship system obtained by analyzing the dependencies of the loop operators in Fig. 2a.

The iteration space is determined by the boundaries of the loop indices and, since nesting of the loop is two, then we consider two-dimensional case in which the variables of the loop vary within $1 \le i \le 6$; $1 \le j \le 10$.

The integer rectangular area bounded by these values represents the iteration space (Figure 3). Each point of the integer space is associated with iteration of the loop with the corresponding values of the iteration variable. Analyzing the program code, one can see that the operator has three inter-iteration dependencies, which can be represented in analytical form as a system of the relations (Figure 2b). In addition, polyhedral model allows not only to represent an arbitrary computational loop as a set of mathematical dependencies, but also to change them and restore a new source code out of them.

```
for i=1 to 6 do
  for j=1 to 10 do
    a(i+j, 3*i+j+3) = a(i+j+1, i+2*j+4)
  endfor
endfor
```

a

```
anti  R1 = {[i,2i] -> [i,2i+1] : 1 <= i <= 4}
anti  R2 = {[i,j] -> [i',i+i'+1] : j = 2i' && 1 <= i < i' <= 5}
flow  R3 = {[i,j] -> [j-i-1,2i] : 2i+2 <= j <= i+7, 10 && 1 <= i}
```

b

Fig. 2. Example of the loop (a) and system of the relations which describes relation graph (b)

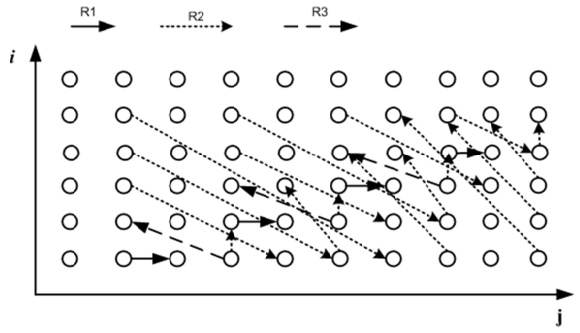


Fig. 3. Graphic form of the iteration space for the loop in the Fig. 2a

Considering the whole set of approaches for the modification of computational loops, such method as tiling should be noted [3]. However, questions about size of the tiles and their shape still remain. Currently, there are the following restrictions: 1) only the rectangular block shape is considered and 2) tiles of iterative space have the same size. In [4] a survey was conducted on the nature of the influence of the block sizes on the execution time of various test programs. Experiments have shown that it is impossible to define a certain pattern of the dependencies between tiles' sizes and execution time.

The source of test programs was Polybench test [5] which contains about three dozen algorithms used in practice from the different data processing domains. 17 package test programs were investigated for the execution time depending on the tile size. To minimize the measurement error five measurements were made for each mode. The largest and the smallest values were not taken into account and the average value among the other three was taken as a valid mean value. The tests were conducted on a personal computer with a quad-core Intel Core i5-4670K processor running on Ubuntu 14.04 LTS operating system.

The examples of the research results are shown in Figs. 4–8.

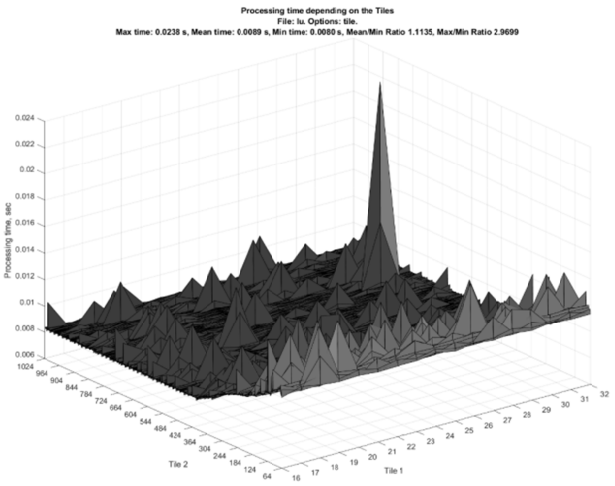


Fig. 4. Example of the execution time on the test program Lu

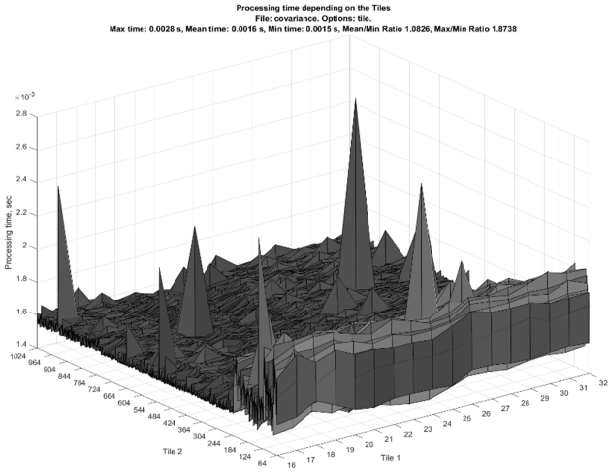


Fig. 5. Example of the execution time on the test program Covariance

As anyone can see, on the one hand, tiles' sizes significantly affect the execution time of the programs. But, on the other hand, there are no obvious relationships between tiles' sizes and program execution time. The best pair of the tiles' sizes for these tests cannot be determined before the experiment. In addition, the examples presented only the two-dimensional case. With an increase in the dimension of the problem, its complexity increases significantly. Therefore, the search for the best tile size can be carried out by successively checking various test tile size pairs. The algorithm itself for the finding of the optimal solution can be built on the basis of the mathematical optimization methods.

Promising in this area are the methods of "swarm intelligence" such as the method of ants, bees, particle digging, as well as genetic algorithms. Next, we consider the particle swarm method in details.

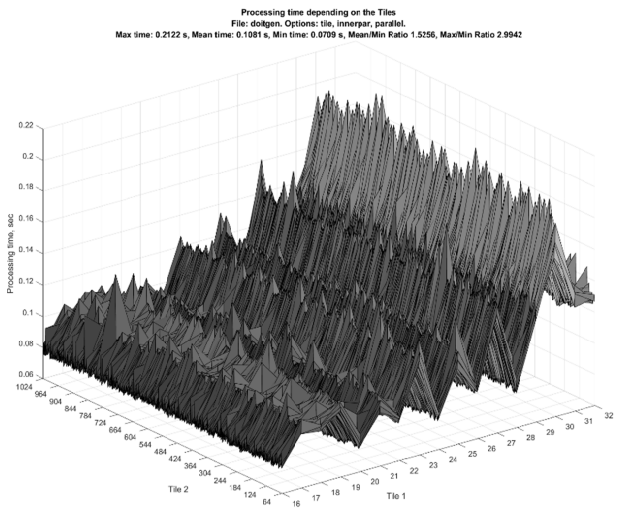


Fig. 6. Example of the execution time on the test program Doitgen

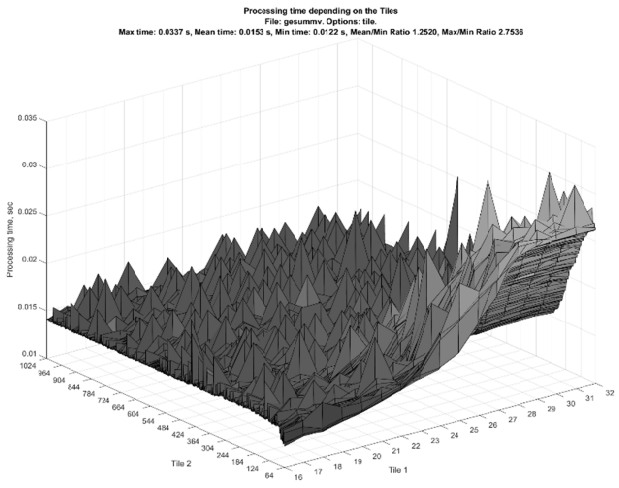


Fig. 7. Example of the execution time on the test program Gesummv

"Swarm Intelligence" in Optimization Problems

Swarm algorithms were used to solve various problems quite intensively, since they are promising in terms of obtaining a global extremum of the target function. In particular, they are widely used for splitting and coloring of graphs, solving the traveling salesman problem, routing traffic, assignment tasks, and other [6]. It is shown that this class of methods demonstrates high efficiency for graph problems. In this case, efficiency must be understood as the possibility of obtaining an optimal solution and also as speed of the obtaining a rational solution. In particular, the method was used to solve the problem of scheduling of mobile brigades servicing ATMs, which is considered as a special case of the NP-difficult task of transport routing [7].

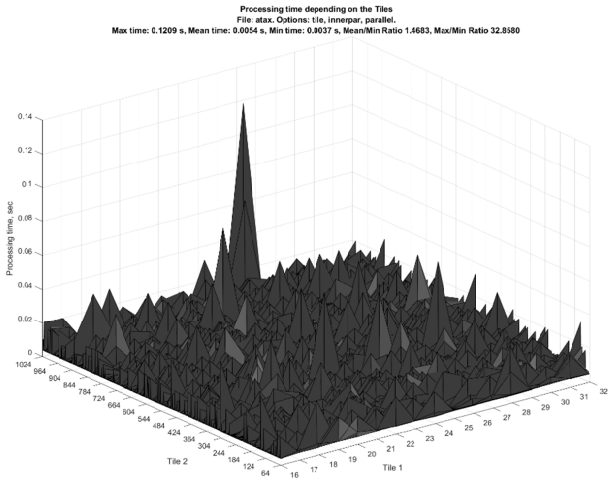


Fig. 8. Example of the execution time on the test program Atax

Considering the problem of partitioning of iteration space into blocks in the tiling method, this is a special case of the graph partitioning problem which is studied in [8]. The task belongs to the class of NP-complete tasks, i.e. to such problems; its solution time depends on the dimension of the inputs. To obtain the exact solution usually a brute force algorithms are used. These include such as breadth-first search, depth-first search, the methods of the dynamic programming, branch and bound method, etc. The main disadvantage of using such methods is that it takes much time to find the results. If the dimension of the problem is n then $n!$ comparisons should be performed. Such methods based on the brute force of the options can be used for tasks that have a small dimension. The usage of heuristic algorithms allows to reduce time to solve the problem but the solution will be approximate. Heuristics limit the search based on certain mathematical laws that reduce the temporal and spatial complexity of the algorithm [9].

One of the modern directions of the obtaining a fairly accurate solution of the problem in the satisfactory time is the usage of the multi-agent methods of the intellectual optimization, which are based on the modeling of the collective intelligence [10]. There are a number of methods that model collective intelligence. For example, such modifications are known as ant and bee algorithms presented in [11, 12]. The optimization using a particle swarm (Particle Swarm Optimization, PSO) is a more generalized search method that is based on the notion of a population and models the behavior of swarm [13, 14].

In [15] the results of using the particle swarm optimization method for the methodology of creating multi-version software (SW), which is based on two main principles, are the following: 1) software consists of modules; and 2) each module has several independent versions that can be combined. The main problem of creating such software is a selection of the optimal version composition of each module in order to satisfy the predetermined restrictions and minimize or maximize certain parameters.

The main statements of the concept of swarm intelligence were introduced by Gerardo Beni and Wang Jing in [16]. A swarm is defined as a decentralized system which consists of a set of simple uniform elements that interact with each other and with the environment to achieve a predetermined goal in accordance with certain rules. The concept of swarm intelligence is built on an additive, synergistic effect, which is manifested when agents are combined into a system. Elements of the swarm are called agents. The model describing the decision of particles in a swarm is based on the position of each particle in the swarm and direction vector. The particle decides on movement based on three factors: its current speed, which causes the particle to continue moving and to explore new regions in the search area; knowledge of your own best state and the best state of the entire swarm or the nearest neighborhood of the particle.

IV. CANONICAL PARTICLE SWARM METHOD

Let *F(X)* be an objective function in the n-dimensional space $\mathbb{R}^n$. Then the global minimization problem is considered as:

$$\min_{X\in R^n} \boldsymbol{F}(X) = \boldsymbol{F}(X^*) \tag{5}$$

Set of particles is denoted by $\boldsymbol{P} = \{P_i, i \in \overline{1..N}\}$, where *N* is a number of particles in swarm or population size. At $t = 0,1,2,\ldots$ coordinates of the particle P_i are determined by the vector $X_{i,t} = (x_{i,t,1}, x_{i,t,2}, \ldots, x_{i,t,n})$, and its speed is a vector $V_{i,t} = (v_{i,t,1}, v_{i,t,2}, \ldots, v_{i,t,n})$. Initial coordinates and velocities of the particles P_i are $X_{i,0} = X_i^0$, $V_{i,0} = V_i^0$, respectively.

$$X_{i,t+1} = X_{i,t} + V_{i,t+1}; \tag{6}$$

$$V_{i,t+1} = a \cdot V_{i,t+1} + U[0,b] \otimes \left(X_{i,t}^{\alpha} - X_{i,t}\right) + U[0,c] \otimes (X_{\beta,t} - X_{i,t}); \tag{7}$$

Here *U[a, b]* is a n-dimensional vector of the pseudo-random numbers, uniformly distributed in the interval [a, b]; $\otimes$ – symbol of component multiplication of vectors; $X_{i,t}^{\alpha}$, is the vector of the particle coordinates P_i with the best (in the sense of (4)) value of the objective function $\Phi(X)$ for the entire time of the search; $X_{\beta,t}$ is a vector of coordinates of the adjacent particle to a given particle with the best value of the objective function *Φ(X)* during the search; a, b, c – free parameters of the algorithm.

During the iterations process vector $X_{i,t}^{\alpha}$ forms the so-called private guide of the particle P_i, and the vector $X_{\beta,t}$ forms a local guide of this particle. Free parameter *a* determines the "inertial" properties of the particles (for a < 1 movement of particles slows down). Recommended value for *a* is 0.7298. In the optimization process a gradual decrease of the coefficient *a* from 0.9 to 0.4 can be effective. At the same time large values of the parameter provide a wide field of the search space and the small ones – precise localization of the minimum of the goal function. The recommended values of the free parameters *b* and *c* are 1.49618 [17].

Thus, the formalized definition of the algorithm of swarm intelligence is determined by a system, as follows [14]:

$$SI = \{S, M, A, P, I, O\}, \tag{8}$$

where *S* – set of agents, *M* – object for the exchange of experience between agents – a certain matrix or vector to which all agents of the swarm have access according to certain rules and which is used in *A*; *A* – rules of creation, behavior, modification of agents; *P* – parameters (heuristic coefficients) used in formulas *A*; , I_1 is an input system, which an objective function is applied to, restrictions – an input for feedback; $O = \{O_1, O_{fb}\}$, O_1 is an output (the best solution found for the problem), O_{fb} is an output for feedback.

Using the swarm algorithm for the problem of optimally chosen tiles' size requires representing of the iteration space as a weighted hypergraph, in which weights are introduced. It corresponds to the computational load of loop operators and graph arcs that correspond to the dependencies in the program. Hypergraph of the iteration space $H = (X, E)$, where $X = \{x_i \mid i = 1,2,\ldots,n\}$ is a set of vertices and $E = \{e_j \mid e_j \subset X, j = 1,2,\ldots,m\}$ is a set of edges. Vertex weight is given by the set $\Phi = \{\varphi_i \mid i = 1,2,\ldots,n\}$ and weight of the edges is set by $\Psi = \{\psi_i \mid i = 1,2,\ldots,n\}$. It is necessary to form *K* nodes, i.e. set *X* is divided into *K* non-empty and non-intersecting subsets X_v, $X = \cup X_v$, $(\forall i,j)[X_i \cap X_j = \emptyset], X_v \neq \emptyset$ [8].

Restrictions are applied to the formed nodes. Using vector $S = \{s_v \mid v = 1,2,\ldots,k\}$ sets a maximum permissible total weight of the vertices assigned to the *v*-th node. Maximum allowable number of vertices assigned to node *v* is defined by vector $N = \{n_v \mid v = 1,2,\ldots,k\}$. Expression (9) is a limit of maximum weight of node and expression (10) is a limit of maximum number of vertices in the node.

$$\sum_{i\in I} \varphi_i \leq s_v, \quad I = \{i \mid x_i \in x_v\}, \quad v = 1,2,\ldots,k. \tag{9}$$

$$|X_v| \leq n_v, v = 1,2,\ldots,k. \tag{10}$$

Multidimensional search space is populated by swarm of particles in heuristic swarm intelligence algorithms. Each particle represents a solution. In our case, this is a solution of the partitioning problem. Process of finding solutions consists in the successive movement of the particles in the search space.

Let's organize an iterative process of moving particles of a swarm and in the finite number of iterations all particles will be concentrated in the extremum. This solution will give the optimal value of tile sizes at the extremum of the objective function, for example, a minimum program execution time.

V. CONCLUSION

The approach to use the particle swarm method to automate the process of finding of the optimal size of tiles of the iteration space of loop operators when the target function reaches the extremum has been considered in the article. The target function can be treated as a program execution time, power consumption of the computing system, size of the program or data memory, etc.

The iteration space is a multidimensional convex polyhedron that is bounded by the values of loop variables. A polyhedron is a directed graph consisting of vertices defined by loop operators. The vertices are connected by arcs in accordance with the dependencies between the operators. To achieve the best performance in parallelizing loops objective function is $F_T = f(S_1, S_2, .., S_n)$, where n is a dimension of loop space. F_T is a multidimensional function that contains local and global extremums.

To determine extremum of function, the usage of optimization methods based on "swarm intelligence" has been regarded. This set of methods is characterized as methods of global optimization which are very likely to find a global extremum. The article describes particle swarm optimization method as the most generalized among many methods of the "swarm intelligence".

Thus, the usage of particle swarm optimization method to find optimal tile size will reduce the time for the automatic parallelization of programs or time for design of the parallel program.

REFERENCES

[1] A. Aho, M. Lam, R. Sethi, J. Ullman *Compilers: Principles, Techniques and Tools*. 2nd ed. Moscow "Williams", p. 1184, ISBN 978-5-8459-1349-4, 2008.

[2] P. Feautrier and C. Lengauer, "The Polyhedron Model" in *Encyclopedia of Parallel Computing*, Springer-Verlag, 2011, pp. pp. 1581–1592.

[3] Xue, J. *Loop Tiling for Parallelism*. Kluwer Academic Publishers. 2000. – p. 256.

[4] S. Sushko, O. Chemerys "Influence of the tiles' sizes operations on the computer program execution time", *Modeling and informational technologies*. vol. 82, pp. 110–117, 2018, (in Ukrainian).

[5] L.- N. Pouchet, (2018, Dec) The polyhedral benchmark suite [Online]. Available: http://web.cse.ohio-state.edu/~ pouchet.2/software/polybench/.

[6] V. Kureichik, A. Kazharov "Using a swarm intellect in solving NP-problems" *News of YUFU. Technical sciences.* – No. 7 (120). pp. 30–37. 2011, (in Russian).

[7] A. Filipova, E.Diaminova, E.Andreeva, E.Laptenok Matrix particle swarm algorithm for support adoption of decision for scheduling service teams. *Proceedings of Sixth All-Russian scientific conference "Informational technology of intellectual support of decision adoption"*, Ufa-Stavropol', pp. 125–128, 2018, (in Russian).

[8] B. Lebedev, V. Lebedev Splitting based on swarm intelligence and genetic evolution. Proceedings of V International Scientific and Practical Conference Integrated Models and Soft Computing in Artificial Intelligence, Kolomna, Fizmatlit, 2009, (in Russian).

[9] J. McConnel, *Fundamentals of modern algorithms*. Moscow Technosfera, 2004, (in Russian).

[10] Engelbrecht A. P. *Fundamentals of Computational Swarm Intelligence.* Chichester, UK, John Wiley & Sons, 2005.

[11] V. Kureichik, A. Kazharov Algorithms of evolutionary swarm intelligence in solving problem of partitioning of graph. Taganrog. Ed. TRTU, 2012, (in Russian).

[12] G. Samigulina, Zh. Masimkanova "Review of modern methods of swarm intelligence for computer molecular design of drugs", *Computer science problems*, No. 2 (31), – pp. 50–61, 2016, (in Russian).

[13] Clerc M. *Particle Swarm Optimization*. ISTE, London, UK, 2006.

[14] P. Matrenin, V. Sekaev "System description of the swarm intelligence algorithms", *Software Engineering, Theoretical and Applied Scientific and Technical Journal*, ISSN 2220-3397, vol. 12, pp. 39–45, 2013, (in Russian).

[15] I. Kovalev, E. Soloviev, D. Kovalev, K. Bahmareva, A. Demish. "Using the particle swarm method to form a multiversion software composition", *Instruments and systems. Management, control, diagnostics*, Moscow, Nauchtechlitizdat, ISSN: 2073-0004, No. 3, – pp. 1–6, 2013, (in Russian).

[16] G. Beni, J. Wang, "Swarm Intelligence in Cellular Robotic Systems," Proceed. NATO Advanced Workshop on Robots and Biological Systems, Tuscany, Italy, June 26–30, 1989.

[17] A. Karpenko, E. Selivestrov, *Overview of the particle swarm methods for the global optimization problem, Scientic ed. of MGTU Baumana* "Science and education", GFBOU VPO "MGTU of Bauman". ISSN 1994–0408. No. 3, 2009, (in Russian).

Aleksander Chemeris. Deputy director of PIMEE NASU. Strong background of elaborating and constructing of high-performance hardware, elaborating and applying program models for the computer simulation as well as developing original programs for the parallel architectures. Good experience in teaching the programming languages. Main scientific interests are in 'Green' computing.

Sergii Sushko. Postgraduate student of Pukhov Institute for Modeling in Energy Engineering, NASU, Kyiv, Ukraine. The author has a great experience of the practical software development and optimization. Area of research: methods of software optimization, automatic source code improvement, energy efficiency of computations, code parallelization and tiling.