ISSN 0361-7688, Programming and Computer Software, 2011, Vol. 37, No. 4, pp. 210–222. © Pleiades Publishing, Ltd., 2011.
Original Russian Text © V.P. Il'in, I.N. Skopin, 2011, published in Programmirovanie, 2011, Vol. 37, No. 4.

Computational Programming Technologies

V. P. Il'in and I. N. Skopin

Institute of Computational Mathematics and Mathematical Geophysics, Siberian Division, Russian Academy of Sciences, pr. akademika Lavrent'eva 6, Novosibirsk, 630090 Russia

Novosibirsk State University, ul. Pirogova 2, Novosibirsk, 630090 Russia

E-mail: ilin@sscc.ru; iskopin@gmail.com

Received October 10, 2010

Abstract—Problems arising in computational programming in connection with the approaching transition to using exaflop hardware are discussed. The discussion is focused on the lifecycle of mathematical modeling and related general problems that are being solved in the course of model development. This area is (and is going to be forever) one of the main consumers of computational resources. The study of its methods and approaches makes it possible to formulate basic directions of assumed progress caused by necessity of formation of computational programming technologies.

DOI: 10.1134/S0361768811040037

1. INTRODUCTION

Until recently, planning of a transition to a new stage of technical provision of computations for mathematical modeling was considered only locally, for separate problems. This was sufficient when the achievement of superhigh performance was viewed as an isolated problem of a particular applied or research task solved in the context of a particular computational environment. Today, the situation is changing. The necessity to solve interdisciplinary problems requires that isolated tasks of particular domains be organized in systems for joint modeling. Such integration often results in systems of partial differential equations, which are difficult to solve by traditional methods.

Barefaced build-up of computational capacities by itself does not change much in what concerns complex solution of mathematical modeling problems. Today, a wide variety of multiprocessor systems positioned as means for high-performance computations are used. They include cluster systems, support of cloudy computing, GRID technologies, powerful graphic accelerators, etc. At the same time, actual performance of such systems is often far from that indicated by the hardware developers and does not ensure current or future needs. According to [1], due to overheads associated with interprocessor exchanges and process synchronization, it does not exceed 10–15% of the peak performance. The reduction in performance is explained by inadequacy of stiff architectures of multiprocessor systems to real computational problems. To overcome this inadequacy, reconfigurable computing systems are currently used, which are built from field-programmable gate arrays (FPGA) [2]. The performance improvement at the expense of growth of the computing power poses the problem of selection of computational environment, and the local adaptation of separate models to each such environment is not only impractical but also impossible.

One of the promising approaches to solving computational problems is to use software-as-a-service (SaaS) provided by the so-called Data Centers, which provide problems with resources with regard to their distribution in the course of the computation process [3]. In this case, the performance improvement is viewed as improvement of efficiency of the Center, which implies that one should not rely on that the model calculation will be carried out with the use of the architecture on which the program was oriented when it was designed. Although the method of adaptation of algorithms and data representations based on low-level optimization does not exhaust itself, its application domain becomes narrower as the complexity of the architectures grows. The low-level optimization is one of the compilation tasks aimed at obtaining an object code that takes into account not only specific features of the architectures but also dynamic properties of the environments where the computation are to be carried out.

All above-mentioned approaches (as well as others) to improving performance at the expense of increasing the computing power illustrate topicality of the transition to technological development of mathematical models, which makes it possible to effectively automate the adaptation of the problem solution to the hardware used. To this end, it is required to develop the following two directions:

• first, algorithmic optimization, the aim of which is to represent the program in the form suitable for automated adaptation to a particular architecture;

• second, construction of large-scale optimized fragments of computations suitable for using in various situations.

The second way is aimed, essentially, at the development of means to support the well-known desire to repeatedly (and multiply) use created algorithms and program modules. It is these modules that require low-level optimization that results in codes that are capable of running in different dynamically varying environments, leaving the programming system only (in the ideal case) the selection of the execution path, which turns out optimal. It is required to develop universal maximally optimized libraries of various algorithms designed for various data representations with support of integration of library modules in composing programs for particular problems, which will make it possible to achieve using almost all capabilities of the computational environment. The creation of premises for gradual transition to solving resource-consuming problems with the use of exaflop hardware, which will replace the currently existing computing systems in a near future, should be considered as a perspective goal [4].

At the same time, methods used currently for programming computational problems (in particular, mathematical programming methods) do not fit the growing architecture complexity and, which is very important, are not designed for simple reuse. This problem makes the researchers look for other approaches and use special tools supporting development of large-scale program complexes, which found application in the field of system and financial programming, the so-called CASE (computer aided software engineering) tools [5]. Unfortunately, such approaches and tools were developed without taking into account specific features of computational problems; therefore, simple transfer of these methods to the new field does no lead to sound results. Computational programming does not turn to what we call technology, really improving efficiency of programmers' labor.

The above-said does not mean, of course, the rejection of the previous experience. Vice versa, it is advisable to take advantage of everything that is suggested for technological support of collective development of system and economical programs, to investigate both popular means of this kind and approaches rejected for various reasons in other that, on the basis of careful studies, determine technologies of computational programming and mathematical modeling with original methods and tools, regulations and agreements, expedients, and design and programming patterns.

Taking into account multidimensional character of the problem of development of such a technology, necessity in significant preliminary investigations, and the subject of the technology to be created, adequate solution of the problem is impossible without fundamental studies, experiments, and verification of the results being obtained. The most suitable approach to solving this problem is an evolutionary way of approbation of the existing methodologies and their development aimed at adaptation to computational programming problems.

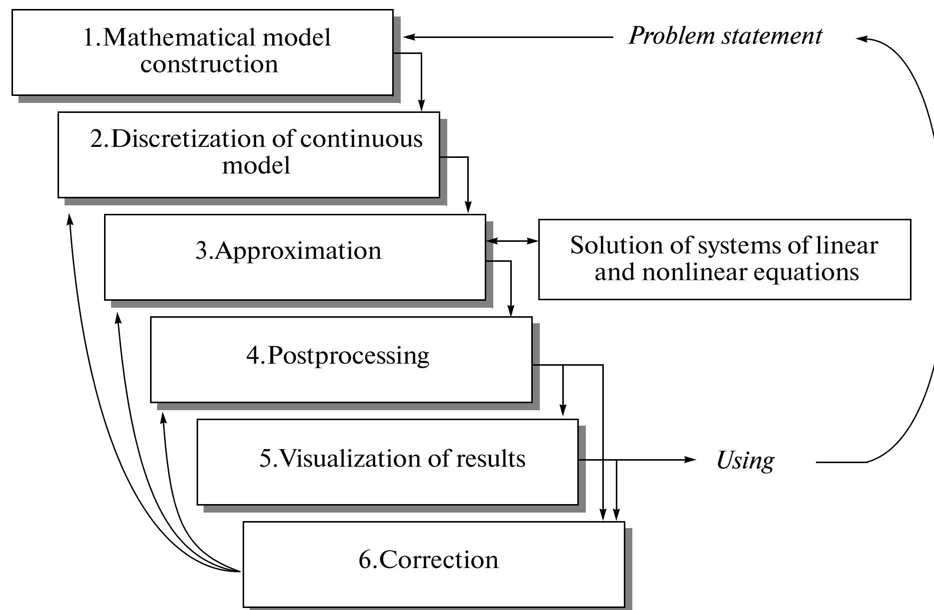
Like in solving any complex problems (in particular, like in developing traditional programming technologies), it is required to divide the problem into a number of relatively independent subproblems. Such decomposition may rely on the lifecycle of mathematical model construction and use in a computational experiment.

An agreement about typical stages will make it possible to systematize solution methods used on each stage and the corresponding algorithms of their implementation fragmentized in accordance with the modular principle, depending on their purposes, approaches used, efficiency, resource consumption, concurrency, etc. [6]. Accordingly, there arise problems of supporting the modeling lifecycle by technological means, which can be characterized as formation of the technological development environment. Essentially, these are possibilities of selection of appropriate means for implementing a well-developed and approved methodical approach to modeling. Advanced libraries (like, for example, [7]) always tend to maximally provide the user with such possibilities, and the only thing that is required for converting them to a technology is to define the so-called operational routes of the user [8] the passage of which (i.e., optimal choice and correct application of appropriate means on each stage) consistently leads to problem solution. However, complexity and diversity of the existing methods (and, those to be developed) of mathematical modeling and interdependency of solutions taken on different stages makes this choice difficult and ambiguous, so that the entire approach becomes a matter of art rather than technology. The approach need to be revised on the basis of complex and comprehensive study of mathematical modeling problems, which will make it possible to create technological development environments that make development of computational technology real and accessible.

In what follows, we discuss both above-mentioned aspects, namely, technological support of the modeling lifecycle and directions of promising studies as a foundation of computational technology.

2. TECHNOLOGICAL SUPPORT OF MODELING LIFECYCLE

In the most general form, the lifecycle of physical process modeling can be represented as the flowchart shown in Fig. 1. The initiation of the modeling originates from the initial *problem statement*, where the need in model construction is fixed and the first stage of the lifecycle is formulated as a mathematical problem. To carry out subsequent calculations, this, as a rule, *continuous, problem is discretized*, which constitutes the content of the second stage. The next stage is *construction of approximations* of the desired functional



Lifecycle of physical process modeling.

relations on the basis of the discretized representation constructed. An important technological tool of modeling on this stage is *solution of systems of linear and nonlinear algebraic equations* describing functional relations. The application of this tool leads to obtaining primary calculation results, which, on the *postprocessing stage*, are reduced to a format suitable for *visualization*, which constitutes the next stage of modeling aimed at submitting the model information obtained for subsequent *using*. Sometimes, a series of calculations are required, for example, when solving inverse problems aimed at finding model parameter values (initial and boundary values, constraints, etc.) for which a desired behavior of the system is achieved (i.e., model optimization aimed at decision making when controlling the computational process). In such cases, as well as when the results turn out unsatisfactory for some other reasons, the model needs *correction*, i.e., elaboration of the discretization, approximation, or postprocessing, which means that the stages need to be organized in a nested loop of stage repetition.

It can be seen from the flowchart that the use of the modeling results are considered to be outside the process of model construction. This is a different kind of activity, which may require modification of the problem statement and transition to a new cycle of modeling. It is important to note also that the construction of a mathematical model is viewed as the base of the entire modeling: if analysis of the data obtained requires modification of functional relations, then a new model need to be constructed.

In the following sections, the outlined stages are discussed from the point of view of model development technology as parts of the general problem of turning computational programming to a technology. For this reason, we do not consider details of each stage concerning particular algorithms but rather focus on general technological aspects typical of any modeling methodology.

2.1. Mathematical Model Construction

The first stage of modeling, as an approach to problem solving, is *selection* or *creation of mathematical methods*. In the majority of cases, mathematical models are described in terms of integral–differential calculus over functions of continuous argument. The direct and inverse problem statements are distinguished: as early as at the stage of selection of mathematical methods, it is required to take into account that, in the former case, a desired functional dependence is sought, whereas, in the latter case, parameters matching a given dependence are to be found, for which an optimality criterion is satisfied. As a result, in the inverse problem statement, methods are selected with regard to multiple repetition of calculations with different sets of parameters; i.e., in essence, solution of the inverse problem is constructed as a series of solutions of the direct problems.

An important aspect of the model construction stage is revealing mathematical properties affecting organization of subsequent calculations. For example, the proof of existence and uniqueness of a solution can guarantee convergence of iterative approximations, which greatly simplifies verification of the results obtained, or solution stability reduces requirements on calculation accuracy support.

The stage completes when the designer approves the selected mathematical model adequately representing an actual problem in the form of a mathematical object and calculation methods used for this object. The stage is verified by a competent expertise, which takes into account not only quality of representing processes under study but also conditions of the real problem (in particular, the way the model data are specified on the user level is to be determined).

For the designer, this is a stage of creative activity not admitting special technologies (except for use of office support means or other general-purpose tools).

2.2. Discretization of Continuous Model

The second stage is called *discretization of the selected model.* It consists in constructing a grid for the calculation region, which is used for replacing an integral–differential model representation by a discrete scheme.

There exist many methods of grid construction meeting various criteria of quality of partitioning the computation region, which affect accuracy and resource consumption of numerical solution. Selection of the method of grid construction may occur difficult for the designer; therefore, technological support here is desirable. Such support is based on analysis of the problem to be solved: mathematical model and parameters describing its particular use, possible variants of specifying computation region, initial and boundary conditions, etc. If this information is not sufficient for complete analysis, the modeling support system may confine its operation to some warnings indicating specific features of the computation region, inconsistency of the discretization and accuracy requirements, and the like. Such warnings will allow the user not to miss important aspects of the model.

Technological tools of discretization include grid generators that perform partitioning of the calculation domain with regard to some performance indexes. This process is controlled by specifying calculation domain features, such as faces, edges, singular points, and the like. The decomposition of the calculation domain into subdomains and grid partitioning of the subdomains should be agreed; i.e., there should be no inconsistencies associated with subsequent approximations of the subdomain into which the calculation domain is partitioned. To make discretization more controllable, various means for editing decomposition of the calculation domain, such as visualization of constructions and coloring of the segments being partitioned, are used.

The stage is completed when the designer is satisfied with the results of the discretization; however, it can be resumed if it will become clear on subsequent stages that this decision was a mistake. Accordingly, the technology should include means for support of resumption, for partial use of results of previous discretizations, and for comparison of results.

2.3. Approximation

The third stage is *selection and implementation of an approximation method*, i.e., transition from original functional relations including an infinite number of degrees of freedom to finite-dimensional equations, inequalities, and recursions. In so doing, the problem, in fact, is reduced to an algebraic form, which is achieved by means of approximate calculation of functions, derivatives, and integrals. Basic approaches to construction of grid relations rely on finite difference, finite volume, and finite element methods; collocations; and spectral methods associated with decomposition into Fourier series. Like on the previous stage, the basic problem associated with the approximation is selection of methods, which can be systematized and classified in terms of various indexes [9]. In terms of technology, the environment for support of development of computational models should ensure selection of a scheme that is adequate to the user need from the point of view of calculation accuracy and performance.

Operations to be performed on the approximation stage greatly depend on the discretization method used; therefore, these two stages are performed in coordination. To be more precise, upon grid generation, the approximation is selected based on an a priori assumption; if results turn out unsatisfactory (which becomes clear on the postprocessing or visualization stage), the discretization stage is repeated with a new a priori assumption.

Technological support of the approximation stage consists in providing the designer with appropriate algorithms, tools for composing calculation scheme, and criteria controlling quality of the results obtained. For such criteria, the order of approximation error, possibility of obtaining an exact numerical solution, rate of convergence to this solution, and stability to perturbations of the initial data and rounding errors can be used. In addition, the tools should support possibilities of returning to the previous stage and comparing variants found. From the point of view of improving calculation accuracy and performance, an automated determination of singularities of the matrix constructed for solving the system of algebraic equations representing functional relations in the model is a necessary element of the technological support of the stage.

The stage includes selection of solvers for systems of grid equations (see the next section) and their uses for organization of calculation of the approximating function, which is considered to be the result of the stage. Since the quality of the solution depends on what solver is selected, the stage may often require several variants of approximation.

2.4. Solution of Systems of Linear and Nonlinear Algebraic Equations

Typically, grid methods for solving differential problems result in large, sparse, and banded matrices. Matrix order N may reach hundreds of millions or billions, and nonzero elements are concentrated in a band of width m near the principal diagonal, with $m \ll N$. The number of nonzero elements in each row usually does not depend on N. Discretized algebraic systems arising from approximations of integral equations (defined on the boundary or in the interior of the calculation domain) are, vice versa, dense but often have special structural properties (for example, these are Toeplitz, quasi-Toeplitz, or block Toeplitz matrices). When solving nonlinear problems, the number of matrices grows significantly, since these problems are usually reduced to systems of linear equations. As a result, complexity of calculations grows. From the above-said, it follows that the main difficulties of solving systems of linear and nonlinear equations are associated with the support of effective manipulation with superlarge matrices of special form.

Computational linear algebra suggests a great number of algorithms for solving problems involving matrices of different types: real and complex, square and rectangular, Hermitian and non-Hermitian, positive definite and sign-indefinite matrices. An optimal choice of an algorithm and the corresponding program is determined by properties of matrices representing calculation subdomains of the model. Possibility of calculation of these properties should be provided as technological support of modeling. The most important part of the modeling, which is responsible for adaptation of calculations to the architecture of the computing system, manifests itself on the level of solving systems of linear equations. The possibility of organizing cluster distribution of calculations, caching, and other specific features of architectures help to reach high performance only theoretically. In order to really increase efficiency of calculations, accurate adjustment of algorithms is required. Mapping of algorithms onto the architecture of computing complexes can be implemented in the framework of specialized libraries. Fast development of such libraries easily-adaptable to new architectures is one of the key problems of development of computational programming technology (see Section 3.6).

Solution of systems of linear algebraic equations (SLAEs) needs special care about calculation accuracy. Ill-conditioned matrices may result in considerable distortions of results and slow convergence of iterations. To compensate this, preconditioning is usually applied, by means of which the original system can be converted to an equivalent one with a better condition number. Again, different preconditioners may be selected, which means that it is required to envision variant operation in order that the user could not face the situation when he is not able to select between one or another strategy.

2.5. Postprocessing, visualization, and correction

The goal of modeling is to use results of computational experiments for performing certain kinds of activity rather than just purposeless solution of mathematical problems. It is necessary to organize separation of required data fragments and to reduce data to formats of their use. Variants of such transformations, which are called postprocessing, may be different: from control signals in automatic control systems to formats of visual representation with possibility of animation of colored images in decision making systems for analysis of results of simulation of modeled processes (sections, isolines, isosurfaces, color coding, plots, etc.). One kind of postprocessing—preparation of information for visual control of grid functions obtained, which is accompanied by model correction—is common for almost any modeling.

Turning postprocessing to technology is achieved by providing the user with a maximally possible set of means for supporting variants of data preparation for subsequent use. Such support may rely on the concept of abstract representation, which is a convenient and fast tool for constructing variants of user data representation. This is the well-known MVC (model, view, controller) approach [10], in the framework of which the development of a program system is based on separation of entities related to processing, control, and visualization of data. In accordance with this approach, the postprocessing stage completes by construction of an abstract representation of calculation results and specification of an abstract control corresponding to schemes of the modeling correction control. Construction of desired visualizations is considered to be a separate stage of modeling.

2.6. General Requirements to Technological Support of Modeling

Formation of technologies for development of mathematical modeling as support of stages of model construction lifecycle is very promising. The above discussion shows that technological modeling requires creation of a specialized toolkit to automate various design operations and development of techniques of these tools use. This is a traditional way that was frequently used for construction of technologies of development of system and applied software. It is quite natural to take advantage of the experience accumulated in this way and to apply it to computational technologies. This should result in considerable growth of labor efficiency in the filed of mathematical modeling.

At the same time, it is clear that the traditional way creates only prerequisites for transition to industrial production of mathematical models. The key problem in the development of computational programming

technologies is to reduce time required for incorporation of promising algorithms, methods, and approaches for support of mathematical modeling that take into account possibilities of new architectures into real practice. Other necessary conditions of turning traditional approaches to the industrial ones are as follows:

• Inclusion in support libraries both programs implementing approved modeling methods assigned to the development stages (and ensuring conditions for their effective use) and means for analyzing the problem being solved and data processed aimed at determination of optimal method selection in each particular case.

• Orientation on adaptation of library means to advanced architectural solutions and use of problem-independent modules ensuring efficient distribution of computational resources and high-performance calculations on the lower common and universal level.

• Openness and extensibility of libraries through both new approaches and methods and architectural solutions (follows from the previous requirements).

An example of the project that is aimed at the implementation of these problems is the development of a specialized base mathematical modeling system BSM [11] carried out currently at the Institute of Computational Mathematics and Mathematical Geophysics, Siberian Division, Russian Academy of Sciences.

3. DIRECTIONS OF STUDIES

Perspectives of the traditional way of development of computational programming technologies should not hide problems associated with the approaching transition to exaflop calculations. As it was already noted, an adequate answer to this challenging transition suggests revision of the existing practices and programming methods and development of fundamentally new approaches, which requires special studies.

This should be a complex study touching all aspects of program development that effectively use perspective hardware. The most topical aspects are listed below.

• It is required to revise approaches and methodology of individual program development.

• The concept of data representation for solving computational problems needs to be extended.

• It is necessary to further develop code optimization methods caused by the need to use nontraditional styles in computational programming.

• It is required to develop approaches to decomposition of computational programs.

• There is a need in new approaches to constructing parallel programs: both to improve the existing parallelization methods and replace them by direct parallel programming.

• There is a need in the development of mathematical libraries of new type that ensure assembly construction and dynamical optimization of the assembled applied programs, as well as their adaptation to the architecture of the computation environment.

• It is required to develop and use methods and instrumental support of collective development of computational program complexes.

• There is a need in studies in the field of system programming and purposeful development of hardware supporting execution of computational programs.

In the following sections, we discuss these issues in more detail and identify key problems needing solutions in each of the above-specified directions.

3.1. Revision of Approaches and Methodologies of Individual Program Development

This is the so-called “programming in small” [12]. An ordinary support of individual methodologies is language means provided for the programmer. We cannot say that, in this field, there are no significant and useful results, which could be used in computational programming. For example, the C++ language standard suggests special classes that help to use the abstract method in solving computational problems. Nevertheless, the use of object-oriented programming in this field did not go further than support of traditional design patterns [14] adjusted to the existing abstractions of computational programming, which are implemented in traditional mathematical libraries.

Alternative methods, such as, for example, use of functional programming means [15], are at the stage of preliminary developments and experiments. Although this field demonstrates certain achievements, these methods are still far from the mass application in practice, to say nothing of the change of paradigms.

The *key problem* here is the lack of certainty about boundaries of adequate applicability of programming styles: the designers say about advantages of an approach suggested but do not mention where these advantages do not work and turn out to be a burden. Another problem is related to the previous one: there is a need in creation of new (and adaptation of the existing) patterns, expedients, and programming methods, in other words, in the development of special programming styles that are adequate to computational problems.

3.2. Extension of the Concept of Data Representation for Solving Computational Problems

The ordinary representation in the form of arrays of real or complex scalars corresponds to mathematical abstraction of vectors and matrices as objects of operation. From this point of view, the use, for example, of sparse matrices and data possessing different features results in abrupt complication of the algorithms, which can be excused only by need of efficiency. This also refers to the concept of accuracy supported in classical mathematics by the concept of approxima-

tion, which is obviously insufficient for programming. From mathematical standpoint, there is a need in the development of approaches that guarantee the desired accuracy of the results obtained [16]. As applied to the programming technologies, libraries should ensure efficient calculations that are adequate to the reality being modeled.

Traditional algorithms are most often constructed as repetition of locally defined fragments, separation of which is a specific feature of one or another approach to problem solution. On the other hand, construction of algorithms where local calculation fragments would be separated with regard to the dynamically changing situation based on global properties of data (which could become a basis for adaptation to the calculation environment architecture) is cumbersome because of lack of development of the required structures. In fact, flexibility of the existing data representations (computational properties of which are always a prerogative of the programmer and are not presented, for example, in the form of accompanying attributes) is not sufficient. The situation is aggravated by the fact that, from the computational standpoint, different structures are adequate to different kinds of processing, and joint support of these structures is usually not provided [17].

The existing methods of domain decomposition—finite volume and finite element methods—demonstrate lack of development of means for operating on grouped data. These methods uniquely separate subdomains for autonomous one-type operation; however, this is not reflected at all on the language level. As a result, advantages of non-interaction are lost upon implementation of the algorithm, and the program complexity grows.

Key problems here are development and implementation of methods for operation on multiply structured data corresponding to computational programming styles and support of actual state of structures useful for subsequent operation. This support should rely on the development of advanced system of data converters ensuring transformation of structures needed for processing and methods of dynamical data analysis ensuring selection of optimal continuation of calculations.

3.3. Development of Optimization Methods and Nontraditional Styles in Computational Programming

The development of programming languages and architectural hardware solutions has been and remains an impulse to improve compilation methods. The need in new approaches, languages, and styles (see Section 3.1) stipulates adaptation of the existing methods (and development of new ones) of efficient implementation of nontraditional languages and, in the first turn, optimization methods for them. Until recently, languages relying on nontraditional calculation models were a matter of only academic interest; therefore, optimization methods for them were not actively developed.

Currently, the situation is changing. Nontraditional styles penetrated into the sphere of computational problems. This especially concerns the functional style of programming, which possesses rather high potential capabilities for computation parallelism and whose expressiveness greatly exceeds that of imperative styles in its application domain. This point is substantiated by examples from paper of Hughes [18], which was published as early as 1989 and has appeared in many copies in press and on the Internet. These are examples of efficient algorithms that hardly could be implemented in terms of imperative programming.

Special means for data organization in a functional program make it possible to achieve high computation performance comparable to that of traditional solutions and to improve it by using new optimization capabilities. An illustration of this is the SaC (Single Assignment C) language [19], in which, by excluding from C everything that prevents functionality, it became possible to get rid of specification of initial and final values of indices and their increments when traversing data regions. Possibilities of optimization of SaC programs execution rely on the absence of side effects, dependence of expression values on the context, and other obstacles preventing functionality in combination with separation of array indexing from data operations. Owing to the use of known optimization techniques, as well as new algorithms, which would be incorrect in the imperative case, the object code gets rid of calculation redundancy and auxiliary arrays.

Nevertheless, functional programming systems are still not capable of competing against programming environments with traditional language compilers. The reasons for this are as follows. So far, functional calculations were not specially supported on the hardware level; therefore, imperative calculations are in an advantageous position. The second reason is related to the first one and consists in the lack of development of optimization methods for a functional calculation model, which is due to the lack of demand of these methods and the existing opinion that this model is not suitable for numerical calculations. Finally, the third reason is traditions that manifest themselves on all levels of computation organization starting from the development of models and algorithms to programming and calculations. These, as well as other, obstacles on the way of development of nontraditional programming styles were mentioned by Backus in his famous Turing lecture as early as 1977 [20].

The traditional model of calculations, which underlies implementations of the existing architectures, is approaching physical limits of performance growth; therefore, we observe the tendency of increasing the number of processors and kernels, and the effi-

cient use of such architectures requires fundamentally new approaches and programming methods. This is just what recognized gooroo Jack Dongarra says time and again [21]. In these circumstances, advantages of nontraditional models (and, in the first turn, functional calculations with their flexible possibilities of parallelism) became apparent, and, as a result, importance of implementations of industrial functional programming systems grows.

The *key problems* of this direction is development of specialized hardware support of nonimperative calculation models and, in the field of programming, mapping of such models into a format adapted to optimization in terms of modern and perspective hardware. In the first turn, this concerns functional languages, whose expressiveness and natural parallelism are rather attractive from the standpoint of solving hard computational problems.

3.4. Development of Approaches to Decomposition of Computational Programs

Currently, the object-oriented approach is the main one in programming. It dominates other (structural, functional, etc.) decomposition methods, demonstrating advantages associated with multiple use of modules adapted to particular situations without loss of functionality [22]. Among the advantages of the approach is compatibility with many programming styles. Having arisen from the need in solving modeling problems and, then, extended to other fields of programming, this approach seems adequate to the tasks of mathematical modeling of physical processes just due to its origination. However, there are two points that need to be taken into account if the object-oriented approach is used as the basis of decomposition in computational programming.

First, computation efficiency should be the subject of special care. The efficiency requirement leads to the solution selection based on criteria that are not related to the object decomposition; therefore, the latter is most often sacrificed. The efficiency is sometimes ensured at the expense of automatic code optimization upon compilation, and this pays off. Nevertheless, information on what data the processing program is going to deal with is not always available upon compilation. Therefore, an attempt is made to use the so-called controlled compilation, for which the user specifies certain optimization options, which determine strategies of code generation. Unfortunately, no significant success was achieved on this way, first, because the relationship between the structure of the processed data and optimization strategies is not evident and, second, because of difficulties associated with selection of options, the number of which in real compilers is too great (may reach several thousand [23]).

The second point is associated with the choice of the basic system of object classes based on which all subsequent program constructions implementing algorithms are performed. The programmers are used to think of the language array structure as a necessary and sufficient abstraction of representing mathematical objects, such as matrices and vectors, in programs. Operations on matrices and vectors are most often specified in terms of variables with indices, which implement access to scalar elements of the array rather than to the array as a whole. This agreement contradicts the commonly accepted concept of objects as active structures, which include data and means (methods) of operation on them, considering an object as an indivisible abstraction. It is difficult to get rid of this “scalar” view of operations on computational structures because the burden of accumulated algorithms and programs based on such understanding is too great and leaves almost no room for alternative abstractions. At the same time, barefaced attempts to write algorithms using vector–matrix notation are certainly not efficient, since result in operating with extremely hard structures. All this is aggravated by efficiency requirements and responses to them in the form of numerous expedients and methods optimizing access to array entries, the support of which is implemented in modern computing systems.

For computational programming, separation of operating with calculation domains from calculations themselves is an important problem of decomposition method development. Autonomous specification of subdomains, neighborhood relations, and rules of data delivery for calculations would make it possible to improve flexibility of calculations, since determination of independent fragments admitting parallel execution is simplified. The SaC language, which was mentioned in the previous section, demonstrates technique of such decomposition and its advantages from the point of view of both expressiveness of algorithm representation and optimization efficiency. It relies on language features originating from functionality of its calculation model.

Separate description of calculation domains and program fragments specifying calculations should be viewed as implementation of the general idea of separating calculation aspects, which, generally, may greatly affect each other. Productivity of this idea is substantiated by its implementation in the approach to organization of stream calculations developed by a research group from the University of Hertfordshire [24]. In this approach, atomic computational components are viewed as autonomous objects operated by the program, which provides control and organizes data streams, taking no care of what particular actions the components actually do. It should be emphasized that the authors of this approach consider development of such programs as self-sufficient activity, which is based on the use of the S-Net language specially developed for this purpose. This language is designed for declarative description of coordination of asynchronously executed components and is supported on the system level.

The idea of separation of the computational process entities aimed at technological development of programs to be executed in parallel on various configurations of computational hardware was used in the fragmented programming approach developed by V. Malyshkin and co-workers [25]. In this case, the focus was placed on autonomous description of computational fragments and data fragments for which dependencies in the form of explicitly specified relations between fragments are established. These dependencies are used for correct determination of control variants and for composing calculation schemes from them with regard to dynamically changing situation. Selection criteria are determined by optimization of the resource distribution and hardware load. In the framework of this approach, the developers managed to reach significant speed-up of performance for some classes of problems. The approach showed its efficiency in solution of computational programming problems [26].

The *key problem* in this direction is adaptation of the existing decomposition methodologies, as well as other successful solutions, to computational programming problems. On this basis, construction of language and system support of program module reuse—a system of classes, tools for operation on fragments and other entities—may be discussed.

3.5. Revision of Approaches to Construction of Parallel Programs

Methods that are currently used for constructing parallel programs tend to the so-called parallelization, i.e., construction of parallel programs from sequential algorithms. In other words, first, a sequential algorithm is constructed (the fact that it can actually be executed by several processors is not taken into account); then, its program is transformed to a parallel version. A more flexible alternative to this is various schemes of construction of algorithms that are free of limitation of one-processor sequential execution. The result of such construction is mapped onto actually available resources, and this is a separate activity not related to composing the algorithm.[1] These activities can be carried out independently one after another. It is also important that the alternative approaches do not need to represent an algorithm in a sequential form if parallel representation of this algorithm is more natural.[2]

Whereas automated parallelization can be excused by the fact that it supports the possibility of using old well-approved programs, the development of new sequential algorithms to be further parallelized for efficient use in modern architectures is a complete anachronism.

The explanation to such an approach is customs and stereotypes of the programmers, who were taught to implement their ideas in sequential programs. Arguments of the apologists of the sequential programming based on the idea that a "human thinks sequentially" do not stand up to criticism: it is not known yet how humans think, but it is clear that putting any limits to the thought process suppresses creative activity. Backus [20] was one of the first who said that customs and stereotypes would become a serious obstacle on the way of development of computational hardware and programming. Today, his words are completely justified.

Stereotypes hamper development of algorithms that, from the very beginning, are adapted best of all for the execution on computing systems with well-developed parallelism and, thus, decelerate further development of such architectures. They have negative effect on the development of the programming languages supporting parallelism. Teaching of informatics and programming, as well as of discrete mathematics, based on these stereotypes reduces quality of education: it teaches to adapt thinking to the existing patterns rather than to break them with the help of new methods. There is a need in revision of the existing modeling methods and finding, at early stages of the development, schemes in these methods that admit parallel formalization, form parallel algorithms, and may result in parallel solutions immediately upon programming.

All this does not mean rejection of the currently used tools of the development of parallel programs. We call to shift accents from program parallelization to direct construction of parallel programs when it is possible. In this regard, the use of Intel® library Threading Building Blocks (TBB) [28], which is written in C++ and designed for support of development of parallel programs in this language, seems to be preferable compared to the orientation to parallelization by means of OpenMP [29] and other systems external to the language in which the program is written. Such position does not contradict the fact that, in the number of cases, parallel construction on the basis of TBB may occur more difficult than construction with the use of OpenMP. This is possible when the case in point is parallelization of a simple sequential code; however, in more complicated cases, the capability of TBB to

[1] Presented in the previous section approaches to the decomposition, which implement separation of calculation aspects, demonstrate productivity of alternative methods of development of parallel programs.

[2] The concept of natural parallelism is not strictly defined. Here, we treat it in a wider sense, as the possibility of formulation of an algorithm without regard to resource constraints (including the number of available processes) for which mapping onto a real hardware configuration presents no difficulties. The problem is said to possess natural parallelism if this mapping can be constructed with accepted performance. An example of such a problem is search of an optimal path between two graph nodes [27]. Unlimitedly parallelized solution of this problem is mapped onto traditional sequential calculation with the help of a quite regular technique.

explicitly specify a calculation scheduling algorithm and to encapsulate the code and data related to the computational stream simplify development and application program code [30, p. 105].

The *key problem* in this direction is development of language means for support of natural decomposition designed for development of parallel algorithms and those that admit effective parallel execution. In this and many other cases, an obstacle is the lack of formulated social infrastructure problem: development of parallel programming methods remains on the level of backyard production and is not provided with technological patterns (the latter should have been taught in the courses of the computational programming discipline at universities).

3.6. Mathematical Libraries Ensuring Assembling of Application Programs and Their Dynamical Optimization

Technologization possibilities were discussed in Section 1 together with the modeling lifecycle stages. This results in the necessity of inclusion of special auxiliary tools into the library, which help to obtain quality solutions. Another aspect of technologization is related to the desire of optimizing the development process. It originates from the idea of variant solution assembly, which includes a motivated set of algorithms selected from the set suggested by the library. A typical feature of such libraries should be use of three optimization levels based on systems of the corresponding concepts, in the framework of which operation with library means is specified.

The system of concepts of the first—*algorithmic*—level is determined by theories used as the base for formulation of algorithms, or, more precisely, algorithm schemes, which specify the library means granted in the most general form. Based on fundamental properties of the operation objects, the designer of the algorithm builds plans of possible variants of the program being assembled. Knowledge of specific features of the processed data and the goals makes it possible to select the desired data transformations from a set of means supported by the library and to arrange them with the aim to determine a constructive and, in this sense, optimal assembly plan.

The second—*language*—level determines elements of the program assembly. The library should include all useful variants of the specified algorithms, which are selected for a real problem based on the analysis of performance requirements, properties of the processed data, and methods of their transfer between the environments where the calculations are performed and data are used. This does not always mean that all variants are implemented. In many cases, it is desirable to construct library elements as parameterized algorithmic structures, which are adapted to the computation conditions at the expense of additional information received from the program being assembled. To match library elements, data converters may be required. In the ideal case, automated transformation of library elements and data structures can be used; however, the user may affect the selection of the approach to be used. The concepts of this level are program interfaces of systems of classes and modules. The corresponding means should be optimized under various computing platforms for practically significant variants of computational environments. Thus, execution of the assembly plan with acceptable performance characteristics is ensured.

The third—*system*—level ensures dialog specification of factors affecting selection of the assembled program and, when possible, automated determination of static and dynamic properties affecting resource allocation for calculations. It suggests possibility of automated (by default) or semi-automated construction; to take into account actual resource demands and constraints, an advanced interactive interface is used. As a result, we have natural combination of components of the program being assembled, including dynamic adaptation in accordance with the solution plan.

The necessity in permanent completion of the library by the tools reflecting modern and promising achievements of computational mathematics gives rise to the requirement of *library openness for extension*. In essence, this requirement is nothing more than specification of algorithmic, program, and system interfaces for library extension modules.

The above-discussed requirements are listed in the project of the specialized base mathematical modeling system BSM, which was mentioned in Section 2.6. It was developed as an extendable library supporting all three optimization levels.

The *key problems* of this direction are matching of solutions on all three levels and creation of an assembly programming toolkit for fast development of libraries easily adaptable to new architectures. Solution matching means possibility of technological transition from one level to another and a matching regulation for modules implementing mathematical models in the framework of the plan of complex solution of real problems by the assembly method. Currently, assembling in the field of mathematical modeling is not supported by means that would make it possible to implement the process as a technology.

3.7. Development of Methods and Instrumental Support of Collective Development of Computational Program Complexes

Modern computational programming is based mainly on individual designer work rather than on collective work. This situation is explained by the existing traditions. In addition, only algorithmic achievements were considered to be significant results in this field, whereas program solutions played an auxiliary role.

This situation started to change radically with the advent of the exaflop hardware, the complexity of pro-

gramming for which exceeds the level of auxiliary support. An expert in algorithms cannot (and should not) go into details of system problems of architecture-oriented optimization and program modularization. Hence, there is a need in cooperation, i.e., group work on projects, and experts in computational mathematics are not ready yet for such work.

In this regard, experience of collective developments of system software products may become a step in the development of technologization of theoretical achievements and methods of mathematical modeling. Recognized methods of distribution of roles in a group; organization of groups; separation of designer groups from the users of large-scale instrumental model complexes; and formation of relations between the designers, users, and customers are applicable to computational programming. The corresponding support tools are also applicable.

Problems of collective development of large-scale software products have been widely discussed in the literature. Among publications on this subject, we note, first of all, one of the first monographs where problems of collective development of software products are systematically discussed. This is the famous book *The Mythical Man–Month* by F. Brooks [31], which is considered to be the number-one bestseller in the programming literature. The monograph [32] is the source of the most complete information about existing methodologies of technological development of software projects. This subject is also discussed in the textbook [8] written by one of the co-authors of this paper. Book *Technology of Programming* by A.N. Terekhov generalizes reach experience of its author in managing large-scale industrial software projects.

The list of publications on methods of collective development of computational program systems will not be complete if we do not mention the so-called agile software development, which currently wins general recognition. This direction consolidated by the manifesto [34] accepted by a group of enthusiasts in 2001 includes various approaches oriented on the existing and new techniques of development support that showed their efficiency in real projects. One of the most advanced approaches among them is called extreme programming [35]. The experience of using this approach and the direction as a whole will be useful in development of computational programming technologies. However, it should be noted that all principles of agile software development cannot be followed when the approach is applied to mathematical modeling. This issue needs special study.

The *key problem* in this direction is incorporation of the existing programming achievements and technologies into practice, which is to be preceded by the stage of adaptation and specialist training. It is also required to develop existing methodologies of collective programming work in the field of computational programming and within frames of special programming styles specific to computational programming.

3.8. Development of Hardware and System Programming Supporting Execution of Computational Programs

A new technology of computational programming will inevitably place problems of purposeful development of hardware and system programming supporting execution of computational programs. In the first turn, this is compiler optimization methods. Today, in this field, methods for improving performance on the lower level [36] are well developed, and there is also some experience of support of specialized calculations, in particular, for graphics [37]. This experience has appeared owing to well-understood bottlenecks arising when solving some problems. The programmers appreciated such support, adapting to hardware capabilities provided for specialized calculations. An illustrative example is the use of support of operation on textures on the level of graphical processors, which gave birth to a new approach called computational programming for graphics accelerators.

The use of new capabilities in a field where they were not supposed to be used is quite natural. It may result in quite unexpected use of specialized means and approaches, which cannot be derived from traditional computation models directly related to numerical calculations. As an example, we refer to work [38], in which, based on capabilities of graphics accelerators, a computation model is defined that makes it possible to replace traditional schemes of mass calculations by combinations of geometric operations and achieve great growth of performance.

We have already noted the necessity in software and hardware support of nontraditional computation models (Section 3.3). In what concerns the hardware, it would be too restrictive to develop only methods that ensure interpretation of new models in the framework of traditional calculations with the help of microprogramming. Although this way yields certain advantages compared to the program interpretation, it does not radically improve performance. It is more promising to seek possibilities that directly implement advantageous features of new models. For example, for functional models, it is more preferable to use hardware with active associative memory. Using the mechanism of coupling of memory elements (tokens) with identical keys, it is possible to run subprograms immediately, as soon as their arguments are ready, escaping the stage of reactions to the corresponding events.[3] However, in order that such capabilities become competitive, calculation optimization methods are required. The use of these methods is not typical of imperative models; therefore, they are not currently well-developed.

In conclusion of the discussion of the subject of optimization for new calculation models, we note one

[3] The details of one of the implementations of this idea can be found in the last section of the memorial collection of papers by V.S. Burtsev [39], where an associative processor developed by the team headed by the author is presented.

of the most critical problems of support of functional calculations. This is implementation of optimal memorization. The case in point is development of schemes capable of reducing the number of expression values stored that ensure that repeated calculation will not required in all contexts of use. It is quite clear that it is useless to store all calculated values, but it is extremely difficult to determine which of them are not required anymore. To overcome this difficulty, the above-mentioned schemes are used.

An ordinary statement of the calculation optimization problem suggests achieving of the maximum efficiency of use of hardware capabilities. The situation with the hardware support of nontraditional models is not exclusion. In this regard, the problem of development of software support for them remains traditional: it is required to ensure architecture-oriented optimization for suggested hardware solutions.

The presented thoughts on the development of calculation support are general and are not directly related to computational programming. Clearly, the above-noted, as well as other similar, solutions should be combined with specialized support adequate to needs of numerical calculations.

The *key problem* in this direction is selection of hardware and software solutions adequate to needs of computational technologies in organization of modeling with the use of expressive language means, which make it possible to reach maximum possible efficiency of calculations.

4. CONCLUSIONS

The problems discussed in this paper do not exhaust the list of all problems that need to be solved on the way of development of computational programming technologies. We considered only those of them that are not sufficiently discussed in publications on transition to exaflop calculations. For this reason, we did not touch, for example, solutions related to organization of interaction and synchronization of streams. They are currently presented in all systems with advanced parallelism, and study of experience of different implementations of the corresponding software—hardware support is necessary for the development of perspective hardware.

We considered problems related to the development of new technologies using example of mathematical modeling of physical processes. The latter is undoubtedly the main consumer of these technologies. However, this does not mean that the other fields needing high-performance calculations should be ignored. Analysis of programming means and organization of efficient calculations in the fields like bioinformatics, simulation modeling, and model-based study of social processes, as well as for other resource-consuming research and engineering developments, should contribute to the determination of means and methods required for the technology discussed. This is necessary in view of conceptual generality of problems to be solved in the course of development of architecture capabilities aimed at exaflop calculations.

We have noted problems of adaptation of the existing approaches to the development of programs that are, in one way or another, related to functional styles of programming and their capabilities from the point of view of computational programming. Clearly, the study should not be confined to only these aspects of computational technologies. We believe that studies in other directions, in particular, those related to sentential and logical languages, development of information systems and databases, and artificial intelligence should also contribute to the development of the technology discussed. There is every reason to further develop the subject discussed.

It can be seen from the discussions of the trends of the development of computational programming technologies that constructive solution of the problems noted in this paper is related to the creation of libraries of new type, which will provide opportunities for adaptation development associated with the optimization of calculations and means for adequate solution of applied problems. Creation and use of such libraries will help to elaborate both particular optimal implementations and methods of work of computational programmers under new conditions. The project of a specialized base system of mathematical modeling BSM, the concepts of which were discussed above, may be viewed as an example of such a solution.

REFERENCES

1. Aladyshev, O.S., Dikarev, N.I., Ovsyannikov, A.P., Telegin, P.N., and Shabanov, B.M., Supercomputer: Application Domain and Performance Requirements, *Izv. Vyssh. Uchebn. Zaved., Elektronika,* 2004, no. 1, pp. 13–17.
2. Kalyaev, I.A., Levin, I.I., Semernikov, E.A., and Shmoilov, V.I., *Rekonfiguriruemye mul'tikonveiernye vychislitel'nye struktury* (Reconfigurable Multipipeline Computational Structures), Rostov-on-Don: YuNTs RAN, 2008.
3. Bennett, K., Layzell, P., Budgen, D., Brereton, P., Macaulay, L., and Munro, M., Service-based Asia-Pacific Software Eng. Conf. (APSEC'00), 2000.
4. Asanovic, K., et al., The Landscape of Parallel Computing Research: A View from Berkley, *Tecnh. Report no. UCB/EECS-2006-183*, EECS Dept., Univ. of California, Berkley, 2006. URL: www.eecs.berkley.edu/Pubs?TechRpts/2006/EECS-2006-183.pdf.
5. Vendrov, A.M., CASE Technologies: Modern Methods and Means for Design of Information Systems, *Portal CITForum*. URL: http://citforum.ru/database/case/ (access date 20.09.2010) (access date 20.09.2010).
6. Il'in, V.P., On Exa-problems of Mathematical Modeling, *Superkomp'yutery i Vysokoproizvoditel'nye Vychisleniya*, 2010, CAD/CAM/CAE Observer #2 (54).

7. Highly Optimized Math Library: Intel® Math Kernel Library, in Intel® Software Network. URL: http://software.intel.com/en-us/intel-mkl (access date 20.09.2010).

8. Skopin, I.N., *Osnovy menedzhmenta programmnykh proektov* (Fundamentals of Software Project Management), Moscow: INTUIT.RU, 2004.

9. Il'in, V.P., Strategies of parallelization in mathematical modeling, *Programmirovanie*, 1999, no. 1, pp. 41–46 [*Programming and Computer Software* (Engl. Transl.), 1999, vol. 25, no. 1, pp. 34–38].

10. Sanderson, S., *Pro ASP.NET MVC Framework*, New York: Springer, 2009.

11. Il'in, V.P., Exa-problems of Mathematical Modeling, *Vestnik YuUrGU*, Seriya "Matematicheskoe modelirovanie i programmirovanie," 2010, vol. 35 (211), no. 6, pp. 29–40.

12. Floyd, R.W., The Paradigms of Programming, *Commun*. ACM, 1979, vol. 22, no. 8, pp. 455–460.

13. Stroustrup, B., *The C++ Programming Language,* Boston: Addison–Wesley, 1997, 3d ed.

14. Gamma, E., Helm, R., Johnson, R., and Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison–Wesley, 1995.

15. Field, A.J. and Harrison, P.G., *Functional Programming*, Addison-Wesley, 1988. Translated under the title *Funktsional'noe programmirovanie,* Moscow: Mir, 1993.

16. Godunov, S.K., Antonov, A.G., Kirilyuk, O.P, and Kostin, V.I., *Garantirovannaya tochnost' resheniya sistem lineinykh uravnenii v evklidovykh prostranstvakh* (Guaranteed Accuracy of Solving Systems of Linear Equations in Euclidean Spaces), Novosibirsk: Nauka, 1992, 2nd ed.

17. Skopin, I.N., Multiple Data Structuring, *Programmirovanie*, 2006, no. 1, pp. 57–72 [*Programming Comput. Software* (Engl. Transl.), 2005, vol. 32, no. 1, pp. 44–55].

18. Hughes, J., Why Functional Programming Matters, *Comput. J.*, 1989, vol. 32, no. 1, pp. 98–107.

19. Sholz, S.-B., Single Assignment C: Efficient Support for High-Level Array Operations in a Functional Setting, *J. Functional Programming*, 2003, vol. 13, no. 6.

20. Backus, J., Can Programming Be Liberated from von Neumann Style? A Functional Style and Its Algebra of Programs, *Commun. ACM,* 1978, vol. 21, no. 8, pp. 613–641.

21. Exaflop Future of Supercomputers. Interview with J. Dongarra, *Superkomp'yutery*, 2010, n0. 1, pp. 21–23.

22. Booch, G., Maksimchuk, R., Engel, M., and Young, B., *Object-Oriented Analysis and Design with Applications*, Addison-Wesley, 2007.

23. Chirtsov, A., Brusentsov, L., Chernyi, I., Grebenkin, S., and Ermolaev, S., Maximization of the Intel Compiler Performance in Iterative Optimization Mode with Feedback, *The 4th Int. Conf. "Software Engineering Conference (Russia)," SEC(R)*, 2008. URL: http://2008.cee-secr.org/en/etc/secr2008_ilya_chernyi_maximizing_intel_compiler_performance.pdf (access date 30.09.2010).

24. Grelck, C., Scholz, S., and Shafarenko, A., A Gentle Introduction to S-Net: Typed Stream Processing and Declarative Coordination of Asynchronous Components, *Parallel Processing Lett.*, 2008, vol. 18, no. 2, pp. 221–237.

25. Malyshkin, V., Assembling of parallel Programs for Large-Scale Numerical Modeling, in *Handbook of Research on Scalable Computing Technologies*, IGI Global, USA, 2010, Ch. 13, pp. 295–311.

26. Arykov, S.B., Asynchronous Programming of Numerical Problems, *Trudy mezhdunarodnoi nauchnoi konferentsii "Parallel'nye vychislitel'nye tekhnologii (PaVT'2009) (Proc. of Int. Sci. Conf. "Parallel Computational Technologies,"* Chelyabinsk, YuUrGU, 2009, pp. 357–363.

27. Nepeivoda, N.N. and Skopin, I.N., *Osnovaniya programmirovaniya* (Fundamentals of Programming), Izhevsk: RKhD, 2003.

28. Intel® Threading Building Blocks 3.0 for Open Source. URL: http://threadingbuildingblocks.org/ (access date 14.02.2011).

29. The OpenMP API Specification for Parallel Programming. URL: http://openmp.org/wp/ (access date 14.02.2011).

30. Kornyakov, K.V., Meerov, I.B., Sidnev, A.A., Sysoev, A.V., and Shishkov, A.V., *Instrumenty parallel'nogo programmirovaniya v sistemakh s obshchei pamyat'yu* (Parallel Programming Tools in Systems with Shared Memory), Gergel', V.P., Ed., Nizhnii Novgorod: NNGU, 2010.

31. Brooks, F.P., *The Mythical Man–Month: Essays on Software Engineering: 20th Anniversary Edition*, Addison-Wesley, 1995.

32. Futrell, R., Schafer, D., and Schafer, L., *Quality Software Project Management*, Prentice Hall, 2002.

33. Terekhov, A.N., *Tekhnologiya programmirovaniya* (Programming Technology), Moscow: Binom, 2003.

34. *Manifesto for Agile Software Development.* URL: http://agilemanifesto.org/ (access date 14.02.2011).

35. Beck, K., *Extreme Programming Explained: Embrace Change*, Addison-Wesley, 1999.

36. Muchnick, S.S., *Advanced Compiler Design and Implementation*, Academic, 1997.

37. Berillo, A., *NVIDIA CUDA*: Nongraphic Calculations on Graphics Processors, 2008. URL: http://www.ixbt.com/video3/cuda-1.shtml (access date 24.11.2010).

38. Il'in, V.P. and Tribis, D.Yu., Geometric Informatics of Continuous Medium Models, *Vychislitel'nye Metody Programmirovanie: Novye Vychislitel'nye Tekhnologii,* 2009, vol. 10.1, pp. 306–313.

39. Nurtsev, V.S., *Parallelizm Vychislitel'nykh protsessov I razvitie arkhitektury SUPEREVM (sbornik statei)* (Parallelism of Computational Processes and Development of Supercomputer Architecture (collection of papers)), Moscow: TORUS, 2006.

SPELL: 1. gooroo