ISSN 0361-7688, Programming and Computer Software, 2016, Vol. 42, No. 1, pp. 5–16. © Pleiades Publishing, Ltd., 2016.
Original Russian Text © V.P. Il'in, I.N. Skopin, 2016, published in Programmirovanie, 2016, Vol. 42, No. 1.

About Performance and Intellectuality of Supercomputer Modeling

V. P. Il'in[a, b] and I. N. Skopin[a, b]

[a] *Institute of Computational Mathematics and Mathematical Geophysics, Siberian Branch, Russian Academy of Sciences, pr. Akademika Lavrent'eva 6, Novosibirsk, 630090 Russia*
[b] *Novosibirsk State University, ul. Pirogova 2, Novosibirsk, 630090 Russia*
e-mail: iskopin@gmail.com

Received February 12, 2014

Abstract—The concept of supercomputer technologies is traditionally related to mapping algorithms onto the computer architecture, which, taking into account the explosive growth of computational capabilities, implies the necessity for an adequate increase in the performance of algorithms and programs. At the same time, it is well known that the rate of building "computer muscles" far exceeds the rate of increasing the labor productivity of software developers, which becomes a bottleneck of computer evolution. The only way to deal with this problem is to automate the construction of models, algorithms, and programs, which directly implies the revolutionary change in the level of artificial intelligence in supercomputer technologies. In this paper, it is from this standpoint that main computational stages of mathematical modeling of various processes and phenomena are discussed, some aspects of high logical complexity of modern high-performance methods for solving "large" applied problems are pointed out, and some intelligent solutions for various modeling problems are proposed.

DOI: 10.1134/S0361768816010047

1. INTRODUCTION

Developing knowledge-intensive software for solving very large mathematical modeling problems on modern post-petascale computers faces two essentially different algorithmic problems. The first problem is traditionally regarded as of paramount importance: mapping algorithms onto the architecture of a multiprocessor computing system (MCS) to maximize the performance or to minimize the computational complexity of the methods used for solving a particular problem. The second problem is associated with the automatic construction of programs and algorithms, which eventually determines the labor productivity of application programmers, the growth rate of which is known to lag far behind the rate of improving computer capabilities. Moreover, as has been noted in the roadmap developed by the experts of the International Exascale Software Project (IESP) [1], the forthcoming entry into the era of exascale computers (with hundreds of millions of cores) implies the transformation of quantity into quality and poses before the world's IT community the challenge problem of developing a new generation of software (X-Stack), which would be able to provide the effective use of extremely large computational resources.

The fundamental solution in this direction is seen primarily as a considerable rise of the intellectuality level in the development of application software, including program packages and software systems. We regard the intellectuality level as a degree of creativity in the work of the domain experts, who are responsible for the conformance between the idea of a software system and actual needs of the users: mathematicians who formalize problems and construct algorithms and programmers who implement a particular software product. For these experts, creativity (in its various aspects) always accompanies their activity when creating a fundamentally new product. At the same time, in contrast to the technology, which theoretically guaranties the timely fabrication of a proper-quality product, the creativity is actually a source of uncertainty and risk. Therefore, the solution of the problems mentioned above calls not for technologies but for the technological support, which provides as maximum automation of routine processes as possible, as well as for the freedom of making creative decisions given all reliable information on all alternatives available. Such a support is based on general software development tools and special-purpose instruments for a particular class of problems.

An example of well-organized technological support is compiler construction. After some works by Knuth had been published [2, 3], which were characterized by quite a high level of intellectuality, it became possible to standardize the software development process and, therefore, to build various tools for support-

ing the construction of the frontend, which represents the initial compiling phases (see, for example, [4]) that do not require any creativity from the developer. However, the same technological elaboration of backend phases, which are related to the architecture-dependent optimization, has not yet been reached, particularly because of the diversity and complexity of possible variants of the optimum code. Constructing this part of the compiler requires high intellectual abilities of the developer.[1]

It should be stressed that the intellectuality, which is related to elaborating good ideas, theories, methods, and algorithms, as well as to introducing them into a programming tool, improves the technological level of the development support, thus ridding a project of the creative work that is performed in advance. This intellectuality depends on the specificity of a particular domain for which application software is developed. Examples of such a dedicated support are software tools that help to overcome logical complexity of activities, such as solving "large" functional systems of equations, handling cumbersome formulas, and constructing algorithms with increased computational accuracy, improved robustness, and other mathematical properties, which characterize the knowledge intensity of application software. In turn, the degree of automation of these activities characterizes the intellectualization of the corresponding computational and information technologies.

Alongside with the domain-specific dedicated support, the technical, system, organizational, and methodological facilities, as well as software development tools that are not problem oriented, also make their contribution to the technological level of project activities. Each of the types of support is progressing with implementation of its achievements, which are based on elaboration of its intelligent products. It is hard to tell which of them makes a greater contribution to the intellectuality level of a project. It should be noted, however, that, when developing a project, the general support creates conditions for the effective use of the dedicated support; this combination improves the technological support of projects and, therefore, the productivity and quality of the programmer's work.

In this paper, we mostly discuss and evaluate the intellectuality of the dedicated support and refer to the other aspects of project technologization only to the extent to which they contribute to the solution of computational programming problems related to mathematical modeling. The paper is organized as follows. Section 2 presents a survey of the processes that are involved, in one way or another, in organizing large-scale computer-aided experiments. The methodological aspects of setting and carrying out computational experiments are addressed in Section 3; these two phases of the experiment are implemented in particular projects in the form of certain stages, including mathematical model development, computational process, and model-based decision making. Section 4 describes some possible ways of automating the algorithm construction in terms of the radical increase in labor productivity when designing the functional and system content of application program packages. Some actual problems of intellectualization and automation of algorithm construction are discussed in conclusion.

2. SETTING AND CARRYING OUT COMPUTATIONAL EXPERIMENTS

Raising the level of dedicated intellectuality is characteristic of individual software development, in which one of the basic quality criteria is the performance of using MCS recourses for computations. Here, one of the most important classes of problems is the description of real processes and phenomena. This class involves interdisciplinary (direct and inverse) initial boundary value problems for systems of differential and integral equations (or their corresponding variational formulations). Such problems are generally formulated in multidimensional computational domains with complex configurations of multiconnected piecewise-smooth boundaries and contrast material properties of mediums, which requires a developed mathematization of modeling.

The technological chains of setting and carrying out large-scale computational experiments involve a great variety of activities. At the top level of consideration, the following stages of a computational project, which characterize both the model development and the computational process, can be distinguished:

• *geometric and functional modeling*: the concretization of a computational domain and its subdomains with equations the properties of which specify certain requirements for using computational schemes;

• *discretization of the modeling problem* that was represented at the previous stage in a continuous form; this stage implies generation of meshes, including adaptive unstructured meshes, which take into account specific characteristics of the formulated problem;

• *high-order approximation* of the original relations by equations on the mesh structure constructed at the previous stage; thus, the problem is algebraized by reducing it to a system of linear and nonlinear equations, which often prove to be cumbersome and ill-conditioned;

• *solving algebraic problems*, including the equations constructed at the approximation stage, by methods that take into account specific features of the matrices involved and capable of solving high-dimen-

[1] In [5], some examples of other software solutions are presented; analysis of these solutions shows that, given an appropriate intellectual elaboration, they can lead to new methods for improving robustness of distributed computing systems.

sional, time-consuming, and ill-conditioned problems;

• *postprocessing*, which is aimed at recasting initial computational results in the form appropriate for the further analysis and use.

When solving inverse problems, an additional stage is required:

• *organization of iterations* for optimization computations, which involves calculating objective functionals and updating boundary conditions, as well as other parameters, in order to return to the previous stages if it is required to continue searching for solutions.

The last two stages imply using the information obtained in the process of modeling:

• *visualization of computational results*, i.e., preparing the results for interactive processing at the final stage;

• *decision making* based on mathematical modeling, which implies validating the results, understanding them, and selecting some final actions.

These stages, as well as special conditions in which they are performed, specify particular requirements for the process of developing the corresponding application software. First of all, mention should be made of the necessity for the system approach to this large-scale problem and for the complex solution of the problems related to the modeling. The continuous evolution of mathematical models and algorithms, which assumes a long life cycle of a software product, requires minimizing the time of implementing its achievements in the product, while the ongoing progress in computer architectures poses the problem of adapting software to new MCS capabilities. The problems of modern and advanced modeling cannot be successfully solved without joint efforts of a great many teams of software developers, as well as end users.

Understanding of the necessity for such cooperation results in new ideas, as well as in attempts to implement them in software systems for technological support of modeling. Examples of such integration projects are OpenFOAM [6] and Distributed Unified Numerical Environment (DUNE) [7]. In this line of projects, special mention should be made of the basic modeling system (BMS) [8], which adopts principles of developing and running software tools for all basic types of problems in mathematical physics: electromagnetism, strain–stress states of solids, hydro- and gas-dynamic flows, multiphase heat-and-mass transfer, etc. This system cannot yet be regarded as an independent software product, but the experience gained in developing its components allows us to conclude that this approach, which enables extending the available methods and adapting them to certain hardware, is rather promising.

3. METHODOLOGICAL ASPECTS OF IMPLEMENTING THE STAGES OF COMPUTATIONAL MODELING

In practice, the above-mentioned stages of setting and carrying out computational experiments are concretized as project development stages; each of these stages needs support, which takes into account both specifics of activities and connections between the stages.

Below, we discuss basic characteristics of the stages, which are peculiar to modeling projects, with the emphasis being placed on the logical complexity of the functional and algorithmic content of the stages. The provisions presented below have been developed based on the conceptual analysis of the BMS, as well as on the analysis of the experience of implementing and running the components of the BMS core [8–13].

3.1. Geometric and Functional Modeling

Solving an initial boundary value problem, or other problem on which the mathematical model is based, begins with specifying initial data with the help of graphical or text tools. This stage of the project corresponds to the stage of geometric and functional modeling; in the general case, the mathematical model being described contains a representation of the multidimensional computational domain and its geometric properties. The domain can be bounded or unbounded and have a multiconnected piecewise-smooth boundary of complex configuration, on various segments of which certain initial and/or boundary conditions are specified. The computational domain can consist of subdomains with different material properties, which involve "their own" systems of differential and/or integral equations with specific terms and coefficients, which, in turn, are given by formulas and/or functional relations or numerically. Altogether, all these are functional objects of the model.

Two important classes of problems are nonstationary problems, in which data vary with time in the process of solving, and nonlinear problems, in which boundaries or coefficients depend on the desired solution and also vary in the process of computations. In addition, inverse problems deserve special consideration, in which initial data contain certain variable parameters the values of which are to be optimized according to additional conditions; these conditions imply minimizing a particular objective functional, which depends on the solution, with linear or nonlinear constraints being imposed on the parameters to be sought.

Obviously, geometric objects can be described in lots of ways. For example, a sphere is uniquely defined by the position of its center and by its radius, as well as by the corresponding equation in polar or Cartesian coordinates, with the first way being more efficient. A truncated cone, cylinder, parallelepiped, and other

standard but more complex figures can be defined similarly. On the other hand, first- or second-order surfaces are defined by coefficients of their equations, while the bodies formed by these surfaces can be identified by finding intersection lines and by executing set-theoretic operations. Finally, the computational domain can be constructed manually on the screen with the aid of standard graphical tools.

In different situations, a particular way of definition can be more or less appropriate; therefore, it is natural to provide for an excessive variety of representations of geometric (and functional) data by enabling their mutual conversion. For example, a great number of geometric formats are available in CAD products [14]. It should also be remembered that geometric modeling must provide tools for executing various operations on objects: shifts, rotations, scaling, copying, etc. A detailed analysis of these technological issues is presented in [10].

It can be seen from the above discussion that the variety of problem formulations and methods for their solution, including methods for defining geometric and functional objects of the model, or, in other words, the intellectuality level of the first stage of mathematical modeling, leaves no hope for a real technology for this stage. And yet, this fact shows the topicality of developing the technological support, which assumes a specific classification of problem types and methods associated with these types, including conditions of their application. It is necessary that verifiable criteria for estimating candidate solutions be always available for the developer of a particular model.

Another aspect of the support is associated with the fact that the analytical work performed at the first stage lays the foundation of all subsequent project activities, including the possibility (oftentimes, the necessity) of checking alternative solutions. Therefore, the support system should be constructed so as to enable identifying operational routes of project development and to ensure easy returns to already completed activities in order to perform them in alternative ways, including returns to the construction of the mathematical model to optimize the construction process, parameter identification, etc. To avoid the risk of chaos in project activities, which can result in lots of errors, to simplify the comparison of alternative constructions of the model, and to enable the use of results of previously performed activities, it is highly important to implement the storage and retrieval of information about actions performed. In other words, the first stage of modeling must involve constructing the project repository with an advanced version control system.

Unfortunately, presently-available development support systems mostly ignore the support of operational routes of project development. As a typical example, we can point to the SALOME system [15], which was designed as a middleware for CAD-CAE. In the course of its evolution, this system has transformed into a platform for mathematical modeling support, which includes some modules for organizing computational experiments. The toolset of this platform is sufficient for solving many problems; however, the absence of mandatory regulations and tools for checking the correctness of their use under particular conditions results in the fact that, when constructing the model, one has to look over the SALOME documentation. And yet, the guidelines contained in this documentation cannot replace automated tools for the dedicated support of proper organization of experiments.

Taking into account the specific character of this stage, the following provisions can be formulated, which characterize the technological support required for this stage:

• in the course of this stage, mathematical properties are determined on which the computational plan depends (on the one hand, the existence and uniqueness of a solution, its stability, etc.; on the other hand, characteristics of the computational domain and its subdomains as objects to be modeled);

• the stage ends with a mathematical model constructed, which represents a real problem in the form of a mathematical object, and with some computational methods for this object found;

• thus, the *geometric data structure* (GDS) and *functional data structure* (FDS) of the model are constructed, which form the foundation of the model for the problem being solved in the project;

• the technological support of this stage is associated with system-wide tools, which provide access to initial information about the problem and make it possible to construct and modify the GDS and FDS (with version control enabled) for the direct use at the other stages (particularly, for version control);

• verification of the results: the competent expertise aimed at validating the quality of reflecting the processes under investigation in terms of the conditions of the modeling problem. This activity is supported by providing access to information about the problem and about its representation in the GDS and FDS.

The first stage of mathematical modeling is associated with making strategic creative decisions, which are crucial for the project; hence, among other computational activities, the intellectuality of the geometric and functional modeling stage is the highest. So, to increase the labor productivity of the developers at this stage and at later stages, a high level of the introduced intellectuality of the general and dedicated technological support is required.

3.2. Discretization of the Initial Boundary Value Problem

Based on the GDS and FDS, the initial continuous formulation is discretized. This development stage consists in constructing an adaptive quasi-structured mesh; this term means that the generated mesh domain can be partitioned into mesh subdomains, each involving various types of cells or finite elements: tetrahedra, prisms, and various polyhedra with some curvilinear edges.

There are lots of publications on mesh generation algorithms, on criteria for determining their quality, and on their software implementations (see a detailed survey in [11]). It should be noted that modern effective methods for solving boundary value problems employ rather complex approaches, such as local condensing, multimesh technologies, and domain decomposition, which require specific support software. The technological tools of this stage include mesh generators and tools for domain partitioning with allowance for some quality criteria. For a particular project, this stage yields a *mesh data structure* (MDS) of the model, which describes (together with the GDS and FDS) all properties of the discretized problem. This structure can be static; i.e., it remains constant during computations, or dynamic, when the mesh is refined or even redefined upon revealing certain properties of the model in the process of computations. The simplest example is reaching a desired accuracy by ascertaining the necessity for condensing the mesh around certain singular points of the computational domain on the basis of a priori and/or a posteriori information.

The dedicated technological support of the discretization stage is reduced to selecting and automating a mesh generation method. It is characterized by the necessity for providing alternative variants of defining the mesh domain and its subdomains taking into account their peculiarities and ways in which initial and boundary conditions are specified. Particularly, for a dynamic MDS, parameterization is required. It is important that the developer be provided with information about these peculiarities on the fly and that the support system be capable of identifying the potential noncompliance of discretization with quality requirements and other errors.

The control of the discretization process is associated with the following opportunities offered to the users:

• specification of features of the computational domain (singular faces, edges, and points);

• consistency of domain decomposition into subdomains and correctness of mesh partitioning of subdomains with subsequent approximation;

• visualization of structures, color highlighting of partitioned segments, and other means for visualizing and editing the image of the computational domain and, therefore, for modifying the GDS and FDS.

The system-wide technological support assumes the possibility of returning to the previous MDS state to correct the actions performed, as well as the possibility of resuming the stage (upon performing subsequent stages) to compare variants of the model with the partial use of the results of previous discretizations. The requirement of such returns is of crucial importance not only for the discretization stage but also for all stages of modeling in any formulation of computational experiments and any organization of project activities. This is one of the key requirements; its implementation in the support system enables the automated comparison of project development alternatives, as well as the check of quality criteria and other important characteristics of the computational experiment. It is natural to consider the operation process that assumes returning to previously performed activities in more general terms. The point is that the advanced modeling support system must support the development of multiversion applications, specifically, enable their adaptation to particular conditions of use. Therefore, the information support of correct returns should be regarded as one of the instruments of the subsystem for version control support. This subsystem is capable of providing conditions for a considerable increase in the labor productivity of both developers and users.

It is evident from the above discussion that the introduced intellectuality has a more important place in discretization than the direct intellectuality, which takes into account certain peculiarities that require the adaptation of standard (or new) mesh generation algorithms. In the case of modeling with the use of a sophisticated support system, this stage proves to be more technologically advanced as compared to the stage of geometric and functional modeling, since it is based on the decisions made at the previous stage, and the use of tools introduces the intellectuality, which involve predeveloped regulations for work activities.

3.3. Approximation of Original Equations

The approximation phase, which is represented in the project by the corresponding stage of model construction, completes the transition from the original continuous problem to its discrete formulation; at this stage, a mesh system of linear or nonlinear algebraic equations (SLAE or SNAE), which generally involves sparse matrices of very high orders (up to 10^8–10^{10}), is constructed. Such matrices are usually compressed: only nonzero elements and the coresponding indices are stored to save memory. There are a variety of approximation approaches, including finite-difference methods, finite-volume methods, finite-element methods, discontinuous Galerkin methods of various

orders, collocation methods, and spectral methods (associated with the Fourier expansion). However, an abstract *algebraic data structure* (ADS) of the model, which is regarded as a conceptual basis of a particular ADS to be approximated, does not depend on this variety. In spite of considerable theoretical distinctions between ADS algorithms, all the ADSs have a remarkable common property: naturally-parallelizable element-by-element methods, which are based on finding local matrices and assembling the global matrix of a problem [12, 16, 17]. It should be noted that the use of high-accuracy algorithms theoretically gives a considerable gain in efficiency but results in cumbersome multipage formulas, the software implementation of which poses a complex technical problem.

Compared to the previous stages, when constructing an ADS, the freedom of choice is confined by previously made project decisions, particularly, by properties of the already generated mesh. The intellectuality of the approximation is most often related to the methods developed in the frameworks of the corresponding theoretical investigations. The key creative problem of this stage is selecting a method for approximation of the original problem that would provide a sufficiently high accuracy and efficiency of computations. It should be noted that this problem is not so simple and has a complex character, since higher-order schemes result in more complex and denser matrix structures, which raises the "cost" of algebraic methods.

The technological support of this stage consists in checking quality criteria: order of errors, potential accuracy of the solution and the rate of convergence to it, stability against perturbations of initial data and against rounding errors, etc. Not all useful criteria can be determined by analyzing the ADS. Some of them, for example, the criterion of maximizing the efficiency of using MCS resources at the next stage of algebraic equation solving, may require additional computational experiments. Generally speaking, this requirement poses a problem, because, for a theoretically admissible computational scheme, it can by no means always be proved that the constructed ADS provides optimally organized parallel computing on available resources. When selecting a scheme, one has to adopt a particular strategy of machine loading and to compare between alternative variants reasoning from the results of further computations, i.e., to organize the return to the definition of the ADS on the basis of preliminary results obtained at subsequent stages of modeling. Thus, checking the approximation quality requires both the support of returning to previous stages and the support of resuming the approximation process upon performing subsequent stages.

The interconnection between the approximation and discretization should be noted. In some projects, the MDS and ADS are constructed simultaneously. Conceptually, such projects use the following scheme: an algorithm and a computational scheme are selected a priori and, then, are iteratively refined. This, however, implies only that the two stages are performed jointly, i.e., the completion of the discretization is not fixed as a check point of the project.

Among the tools of the approximation stage, automated tools for determining properties of the generated matrix play a crucial role. The use of such dedicated support tools considerably improves accuracy and efficiency of computations and, in particular, makes it possible to get rid of solving a more complicated eigenvalue problem. These tools also increase the labor productivity of the developers by ridding them of routine operations.

3.4. Solution of Algebraic Problems

Methodologically, the stage related to solving systems of linear or nonlinear equations can be regarded as a process of finding unknown ADS variables. Project developers often have to deal with very large SLAEs, the solution of which is the most resource consuming stage in the process of modeling, since the amount of computation grows nonlinearly with increasing number of degrees of freedom. It is at this stage that the adopted parallel computing strategy is fixed to maximize the efficiency of using MCS resources.

This direction is one of the rapidly developing branches of computational mathematics. Here, the most effective approaches are based on parallel methods of domain decomposition and on preconditioned iterative processes in Krylov subspaces (see the references in [13]). To illustrate the variety of algorithms used for solving SLAEs, we can refer to the problems involving various types of matrices: real and complex, square and rectangular, Hermitian and non-Hermitian, positive-definite and indefinite, etc. In these problems, one has to deal not only with computational complexity but also with huge amounts of transferred data, so certain problems of computational algebra are presently associated with data-intensive computing.

A broad spectrum of approaches potentially applicable for solving systems of equations, which is regarded as the intellectuality introduced into the project, becomes much more involved because model developers have to deal with real computational resources. They have to make decisions about applicability of a particular method in the context of available MCSs, to take into account perspectives of their development, and to make modeling systems capable of adapting to various hardware configurations with as full and efficient use of resources as possible. Because of the steadily increasing complexity of computer hardware, including the development of new architectures, the adaptability is an "endless" problem to be permanently solved throughout the history of model-

ing. This, however, is a shared problem for computational programming as a whole.

For developers of a particular project, the main problem of this stage involves the algorithm selection and the construction of programs implementing the computational scheme fixed at the previous stages by using libraries of the modeling support system and by taking into account the architectures used for the computational experiment. This problem also assumes the specification of requirements for computational resources needed for a particular experiment. Today, the developers have at their disposal lots of libraries for solving SLAEs, each having its own pros and cons in terms of functionality and other characteristics. When deciding which libraries are to be used under particular conditions, it is rather useful to experiment with various SLAE collections, which are available in the Internet.

The ambiguity of these problems suggests that their technologization should be associated with the support of creative processes, which are released from routine and are accompanied by the protection against making wrong decisions. Since the basic requirement for this stage is maximizing the efficiency of computations, when using the support, it is necessary to avoid situations in which the developer cannot prefer any particular strategy for local selection of variants among available alternatives. When designing such a support, it is required to provide the possibility of tuning algorithms of algebraic libraries to specific characteristics of problems and hardware, as well as to take into account questions concerning computational accuracy, selection of preconditioners for computation speedup, and other aspects of real projects.

The above considerations substantiate the assertion concerning the high intellectuality of technological support systems used at the stage of algebraic problem solving. Here, the share of the introduced intellectuality—methods, algorithms, etc.—is higher than that of the intellectuality associated with the specific characteristics of the project, which call for some creativity to select a solution method. Such specificity includes particular characteristics of the computational scheme and model matrix, as well as MCS resources allocated to the computational modeling experiment.

3.5. Postprocessing

Investigation of complex phenomena or processes by means of modeling is a long-term process involving intensive human–computer interaction. This requires supporting at least two types of activity, which characterize the stages of development and use of models: processing of computational results and their subsequent representation in a usable form. The first type of activity—*postprocessing*—prepares data obtained at the previous stages for the second type called *visualization*. Without postprocessing and visualization, the on-line analysis and control of the computational process are impossible.

The postprocessing stage is aimed at reducing the modeling results obtained at the approximation and equation solving stages (i.e., the MDS and ADS that contain the data obtained as a result of computations) to the form required for analysis and decision making.

The postprocessing can be concretized in various ways, but developers of the model are usually aware of the structure required. For example, in automated process control systems, control signals are generated that stimulate the activity of a process to be controlled. Given a high level of automation, there can be no real user of the model, i.e., the user can take no part in decision making. The situation is reversed when modeling is used for research and analysis. Here, various formats for visual representation of multidimensional fields are required: isolines, isosurfaces, gradient characteristics, plots, etc. (with or without animation). The number of useful representation types can be considerable, and, which is quite important, the choice of particular representation formats is beyond the scope of model development. Hence, it is reasonable to regard the postprocessing as a process of obtaining a buffer representation, which assumes, first, the check of modeling quality criteria (which should be simple to set and easy to perform by the user) and, second, the fullest set of tools for visual representation of information about the model. Moreover, when solving inverse problems, the automatic organization of computational iterations is required (see the following subsection).

The tools for computational data representation can be characterized as a possibility for constructing various *particular data structures of modeling results* (PDSMRs). The role of postprocessing is to create an *abstract data structure of modeling results* (ADSMR), which is regarded as a basis for implementing PDSMRs required for the project developers or users. Taking into account a great variety of PDSMRs, it is reasonable to standardize the ADSMR format for the technological support of postprocessing. An important requirement for this standard is defining abstract control of operations on the modeling results, for which a user interface is designed at the visualization stage. This makes it possible to organize a technological support in accordance with the well-known model–view–controller (MVC) design pattern [18]: postprocessing is responsible for the *model* and *control*, while visualization is responsible for the *view*. As a result, the level of the introduced intellectuality for particular projects is raised.

The technological connection between the postprocessing stage and the discretization stage should be noted, which is due to the requirement of matching mesh representations of model data. The end result of postprocessing the ADSMR forms a basis for demonstrating the computational data to the user. Moreover,

the same structure can be used to visualize data for project developers, who modify the MDS on the fly to improve the model. Therefore, it is reasonable to construct tools for ADSMR visualization so that to enable the use of the same toolset for discretization and approximation. In other words, the visual representations of the MDS and ADS in the project should be regarded as variants of a PDSMR, which are adapted to be used by the developers.

3.6. Organization of Iterations for Optimization Computations

An important stage of the modeling process is the solution of inverse problems on the basis of optimization principles, which consist in the conditional minimization of a predescribed objective functional. This approach assumes solving a sequence of direct problems with parameterized data. In recent decades, a great number of effective algorithms for finding desired parameter values (interior point method, sequential quadratic programming, method of confidence intervals, etc.) have been developed, which are now being actively put into practice.

The organization of the iterative computational process is the key condition for solving inverse problems. However, the need for returning to previous stages also arises when solving direct problems. The point is that an obtained solution may prove, for various reasons, to be unsatisfactory, so, in order to obtain useful results, the correction of previously performed activities is required, which is organized as iterative returning to previous stages. The optimization of algorithms for solving direct problems is essentially an inverse problem, which implies finding desired parameters of the method used. As noted above when discussing approximation, such an iterative process is organized, for example, when constructing the ADS and computational scheme, which use MCS resources to the maximum extent possible (see Section 3.3). The difference between the two types of iterativeness is that, first, they have different termination criteria and, second, for inverse problems, both criterion checking and returning to the previous stages are automated (for direct problems, the decision concerning the return requires a special analysis to substantiate the necessity for correction of previously performed activities).

The technological support of both types of iterativeness is reduced to standard (common for programming systems) version control tools. These tools enable easy returning to any previous state and make it possible to compare and use previous versions of programs and data. As noted above, such an operating mode must be provided for in the project by default. So, the intellectuality of the iterativeness support proves to be system-wide and introduced. The dedicated part of this support is required only for inverse problems due to the automation of returns to previous stages upon calculating the target functional.

3.7. Visualization of Computational Results

Dividing the process of computational data preparation for external use into the abstract and concrete parts makes it possible to considerably simplify the presentation of computational results to the user. In other words, postprocessing is responsible for data mining, while visualization is responsible for data viewing (demonstration of data for the purpose of analysis or process control). Data viewing and control activities are associated, first, with graphical and interfacing tools and, second, with data transfer between the MCS and the user's workstation. These are two different problems. The first problem is solved at the visualization stage. As for the second problem, it should be considered in a broader context, since the data communication between the MCS and workstations should be provided not only for users but also for model developers. Moreover, it is sufficient for users to operate with a PDSMR only, while developers need access to all model data structures and to all relevant information throughout the development of the project. The developers have to deal with project documentation, tests and test results, various versions of basic structures, etc.; it is not necessary and, in most cases, not desirable or even impossible to store a lot of that on workstations.

Viewing of data, which are represented in the ADSMR and are mediated via a PDSMR, to the users depends on various reasons (user needs). Using a developer's PDSMR (even simplified to a desired level) for this purpose requires special considerations. In addition to factors directly associated with the content of modeling, user needs also depend on the specific nature of the activity in which the information obtained from the model is used. It is this specificity that determines which tools are required for operating with model data and which interfaces are needed for these tools. To satisfy user needs as fully as possible, it is reasonable to develop these tools as programs written in domain specific languages (DSLs) [19], which take into account the specificity of particular fields of interest. The choice of a particular language depends on the general abstract model of computations and particular representations of this model, which are specific to each user community. As applied to computational modeling, the ADSMR is used as a basis for the abstract model, while particular representations are based on corresponding PDSMRs. As for computational programming, this approach is employed, for example, in the OpenFOAM system [6]. Designing certain DSL families required in the computational modeling project is regarded as a dedicated intellectuality, while the support environment for developing their programming systems is regarded as an introduced intellectuality of the project.

3.8. Decision Making

Model construction is always aimed at providing the user (researcher, engineer, or other specialist) with some information required for decision making. This information is generated by model-based computations and is used in a particular problem domain. Decision making is an activity which is not performed by model developers, so it should be regarded as a post-project activity, which, strictly speaking, is not a project stage. Nevertheless, at least some project developers take part in this activity; their role is related not to the creation of a user product but to the support and maintenance of a final model, to the training of users with different skill levels and research orientations, and to the development of the basic modeling support system. These related activities may require developing special visions of software, which implement new models and algorithms taking into account cognitive and ontological aspects of the evolution of the computing and information environment.

The post-project activity and its intellectuality are beyond the scope of this paper. However, its obvious connection with the modeling problem deserves special consideration. This connection goes beyond the use of computational data for selecting a solution among available candidates and, first of all, consists in the cooperation between users specialized in certain application domains and model developers. Such cooperation is mostly aimed at the model validation (i.e., checking the agreement between modeling results and realities of an application domain) and at the analysis of information extracted from computational data. The inadequacy of the results points to the necessity for correcting the model, parameter relations, requirements for computations, etc. The further analysis shows the limits of applicability of the results for decision making. In the process of model construction, domain specialists act as consulting experts for the developers; in the post-project phase, the developers become such experts, who consult domain specialists on the use of modeling results and estimate the feasibility of suggestions concerning the improvement of the model, including the corresponding resource requirements.

The cooperation between the developers and users of the model suggests that it would be reasonable to use general tools in the process of model-based decision making, which are available in the basic toolset of a modeling environment (as applied to the BMS, such tools are discussed in [8–12]). The generality of the technological support increases the labor productivity and reduces costs associated with analysis and decision making owing to the introduced intellectualization of the corresponding processes.

4. SURVEY OF ARCHITECTURAL REQUIREMENTS FOR THE BASIC MODELING SYSTEM

Software implementation of the methodological stages described above and their representation in the technological support required for particular project stages form a *kernel* of the BMS, which is directly responsible for the functionality of models and for the labor productivity of the developers. It should be stressed that the BMS kernel consists of almost *independent units*, i.e., self-sufficient subsystems, which interact only via the above-mentioned data structures related to the model construction (GDS, FDS, MDS, and ADS) and to the effective use of modeling results (ADSMR and user-adapted PDSMR). Each individual unit and the kernel as a whole offer an integrated tool environment, which satisfies all technological requirements for the extensibility of the set of available models and algorithms, for the adaptability to the BMS architecture, and for the inclusion and use of external software. The units have a library (modular) structure with standard internal interfaces, which make it possible to assemble algorithmic components and to control correctness of their execution.

These functions actually consist in controlling the operation of the modeling system and constitute the system content of the BMS, which is aimed at the general version control support of the whole toolset, software applications, and collections of applications for solving particular problems. Such an architectural solution, which is hereinafter referred to as the *version assembly*, is similar to the Lego construction kit, which offers a set of standardized pieces that allow for construction of a variety of stable models.

The version assembly is aimed at overcoming the eternal contradiction between universality and effectiveness. Of course, it assumes redundancy of the functional content, when the library contains several algorithms for solving the same subproblem, which are automatically selected according to certain features specified in the configurator. Similar variety makes it possible to support multilingual model implementations and adaptability to various operating systems. These undoubtedly positive qualities of the BMS, however, can create certain difficulties for the developers who have insufficient knowledge of its components. The solution of this problem is that the environment must present the full set of tools available for correct operation, along with admissibility conditions of their use. It would be a mistake to offer tools without indicating the activity to which the tools are applicable. Such a mistake leads to the decrease in robustness of BMS applications.[2]

[2] The increase of robustness is a special problem of software development, which deserves special consideration; some approaches to solving this problem can be found in [5].

To implement the version assembly, a special regulation concerning components of the technological support environment is required. Such regulations were developed and successfully used for various branches of system programming. The most widespread of them is the component object model (COM) [20], which supports the development of interacting components in lots of Microsoft products (particularly, in Windows family operating systems). The common object request broker architecture (CORBA) [22] supports the development of distributed computing applications [21]. For computational programming, the common component architecture (CCA) [23] was proposed, which supports massive parallel processing. Unfortunately, these and other regulations not fully satisfy the requirements for supporting the effective implementation of high-performance computing, so their direct use in the basic modeling support environment makes no sense. The developers of the SPARSKIT system [24] have proposed some approaches to overcome this drawback (see [25, 26]).

In contrast to well-known commercial products, such as ANSYS, NASTRAN, etc. (the related information can easily be found in the Internet), the BMS, a large corporate product, is designed and implemented in the framework of the open source paradigm, which, in a wider context, corresponds to the concept of open innovations [27]. Of course, such a policy is not in contradiction with the development of special-purpose closed source packages on the basis of open tools.

Since the BMS is principally oriented to the high-performance solution of very-large-scale interdisciplinary problems, it is reasonable to use this system in cloud data centers (this technology is now being widely accepted) taking into account the extending range of computer services (software as a service, SaaS). The BMS itself becomes a multiuser system, which inevitably affects its architecture. Particularly, in accordance with the IESP paradigm, this system is being developed with no formal software constraints imposed on the number of problem orders and on the number of processors and computational nodes (cores).

CONCLUSIONS

Now, returning to the central problem of this paper—intellectualization and automation of algorithm construction—we want to cite a figurative slogan from [19]: the transition from paleo-informatics to neo-informatics. This slogan reflects some evolutionary trends in application software (particularly, in computational modeling), which are associated with the development of requirements for scenario formulation and for computational experimentation. Applications that can be attributed to neo-informatics are inevitably associated with multiple transformations of data, as well as with the variety of tools for operating with data. Therefore, for the efficient technological support of developing these applications, a system for constructing domain-specific languages is actually required, which reflects this variety. Above, we have already spoken about the DSL as a method that provides the users with tools for operating with model data. It is easy to see that the diversity of data structures and their modifications to deal with is characteristic of model construction problems. So, it is very tempting to use the idea of DSL for supporting the activities associated with the development of the BMS. Hence, the concept of model implementation inevitably changes: a so-called mogram (model program) rather than a program, which only transforms input data into output data, is required. The term "mogram" was introduced by Kleppe in [19] to describe software that allows for multiple and multivariate experiments with an essentially unified model, which employ general information support and a toolkit required for operating with data. In terms of the data structures introduced above, a mogram is a system of GDS, FDS, MDS, and ADS, as well as the ADSMR and PDSMR, structures.

A serious problem of the transition to neo-informatics is the necessity to maximize the applicability of previously developed software. In the case of system programming, this problem is solved in many respects by using object-oriented design and other modern solutions; in the case of computational programming and mathematical modeling with their increased requirements for computational efficiency, one has to search for special approaches, which support multiple use of presently-available programs, including FORTRAN programs written in the days of paleo-informatics. This objective is set in the scientific interface definition language (SILD) project for support of the so-called scientific computing. Here, the Babel multilingual system is proposed, along with the corresponding toolkit, which makes it possible to adapt old software to modern MCS [28]. Unfortunately, despite the considerable advance in this direction (see, for example, [29]), the technological progress in solving the problem of reuse, which would satisfy the increasing requirements for efficiency, has not yet been made.

The problem of developing domain-specific software, as well as examples of solving this problem, has been known for many decades [30], and the corresponding multifunctional systems such as MAPLE, MATLAB, etc. are now being widely used. Unfortunately, embedding such "monsters" into application software packages poses a nontrivial and most often unsolvable technical problem. Hence, the practice of developing small special-purpose subsystems for automatic implementation of specific applications (e.g., for analytical computations) has become globally

accepted. As an example directly associated with mathematical modeling, we can point to the construction of Nedelec's finite-element vector basis functions with orders from first to fourth inclusively, which is implemented in the HELMHOLTZ 3D software package [31]; based on these functions, modules for finding local matrices and for electromagnetic field postprocessing are constructed. The following example substantiates the sufficient credibility and effectiveness of this approach: for cumbersome formulas (which occupy up to ten text pages), automated analytic transformations yield about 9 MB of the C++ code. It should be stressed that, given the further evolution of finite-element tools, such a subsystem will increase the labor productivity of the application programmer by orders of magnitude.

The technological problems considered above form a broad field of activity for system programmers: analytic geometry algorithms, decomposition of a mesh domain into balanced subdomains with specified degrees of intersections, construction and optimization of computational algebra methods, special tools for working with graph structures, etc.

In conclusion, it should be underlined that the ongoing global transition from the simplest algorithms to the most efficient ones leads to a considerable increase in their logical complexity, so only active intellectual innovations, which provide a real possibility of using integrated environments (which offer, along with various toolkits, basic component modules as building blocks), can accelerate the implementation of the latter. Essentially, this implies the transformation of the craft-like approach to modeling into the advanced, complex, expandable, adaptive, and technologically supported activity.

ACKNOWLEDGMENTS

This work was supported by the Russian Science Foundation, project no. 14-07-0048.5.

REFERENCES

1. International Exascale Software Project (IESP). http://www.exascale.org/iesp.
2. Knuth, D.E., Top down syntax analysis, *Acta Inf.*, 1971, vol. 1, no. 2, pp. 79–110.
3. Knuth, D., On the translation of languages from left to right, in *Information and Control*, 1965, pp. 607–639.
4. Levine, J., Mason, T., and Brown, D., *Lex & Yacc*, O'Reilly Media, 1992, 2nd ed.
5. Skopin, I.N., An approach to the construction of robust systems of interacting processes, *Parallel Programming: Practical Aspects, Models, and Current Limitations*, Tarkov, M.S., Ed., 2014.
6. OpenFOAM open source CFD toolbox. http://www.openfoam.com.
7. Distributed and unified numerics environment (DUNE). http://www.dune-project.org.
8. Il'in, V.P., Strategies and tactics of "beyond-the-clouds" mathematical modeling, *Trudy mezhdunarodnoi konferencii PAVT 2014* (Proc. Int. Conf. PAVT 2014), Chelyabinsk: South Ural State Univ., 2014, pp. 99–107.
9. Il'in, V.P. and Skopin, I.N., Computational programming technologies, *Program. Comput. Software*, 2011, vol. 37, no. 4, pp. 210–222.
10. Golubeva, L.A., Il'in, V.P., and Kozyrev, A.N., On software technologies in geometric aspects of mathematical modeling, *Vestn. Novosib. Gos. Univ., Ser. Inf. Tekhnol.*, 2012, vol. 10, no. 2, pp. 25–33.
11. Il'in, V.P., DELAUNAY: A technological mesh generation environment, *Sib. Zh. Ind. Mat.*, 2013, vol. 16, no. 2(54), pp. 83–97.
12. Butyugin, D.S. and Il'in, V.P., CHEBYSHEV: Principles of automation of algorithm construction in an integrated environment for mesh approximations of initial boundary value problems, *Trudy mezhdunarodnoi konferencii PAVT'2014* (Proc. Int. Conf. PAVT 2014), Chelyabinsk: South Ural State Univ., 2014, pp. 42–50.
13. Butyugin, D.S., Gur'eva, Ya.L., Il'in, V.P., Perevozkin, D.V., Petukhov, A.V., and Skopin, I.N., Functionality and technologies of algebraic solvers in the Krylov library, *Vestn. Yuzhno-Ural. Gos. Univ., Ser. Vychisl. Mat. Inf.*, 2013, vol. 2, no. 3, pp. 92–105.
14. Malyukh, V.N., *Vvedenie v sovremennye SAPR* (Introduction to Modern CAD Systems), Moscow: DMK, 2010.
15. SALOME open source integration platform for numerical simulation. http://www.salome-platform.org.
16. Il'in, V.P., *Metody konechnykh raznostei i konechnykh ob''emov dlya ellipticheskikh uravnenii* (Finite-Difference and Finite-Volume Methods for Elliptic Equations), Novosibirsk: Inst. Comput. Math. Math. Geophys. Sib. Branch Russ. Acad. Sci., 2001.
17. Il'in, V.P., *Metody i tekhnologii konechnykh elementov* (Finite-Element Methods and Technologies), Novosibirsk: Inst. Comput. Math. Math. Geophys. Sib. Branch Russ. Acad. Sci., 2007.
18. Rogachev, S., Generalized model-view-controller. http://rsdn.ru/article/patterns/generic-mvc.xml.
19. Kleppe, A., *Software Language Engineering: Creating Domain-Specific Language Using Metamodels*, New York: Addison–Wesley, 2008.
20. Oberg, R.J., *Understanding and Programming COM+*, Prentice Hall, 1999.
21. Tel, G., *Introduction to Distributed Algorithms*, Cambridge University Press, 2001, 2nd ed.
22. Common Object Request Broker Architecture (CORBA). http://www.corba.org.
23. Common Component Architecture Forum. http://www.cca-forum.org.
24. SPARSKIT: A basic toolkit for sparse matrix computations (version 2). http://www-users.cs.umn.edu/ ~saad /software/SPARSKIT/index.html.
25. Malony, A., Shende, S., et al., Performance technology for parallel and distributed component software. http://people.cs.uchicago.edu/ntrebon/docs/gridperf02.pdf.

26. Alexeev, Yu., Allan, B.A., et al., Component-based software for high-performance scientific computing. http://iopscience.iop.org/1742-6596/16/1/073/pdf/1742-6596_16_1_073.pdf.

27. Chesbrough, H.W., *Open Innovation: The New Imperative for Creating and Profiting from Technology*, Harvard Business Press, 2006.

28. Babel high-performance language interoperability tool. http://computation.llnl.gov/casc/components/index.html#page=home.

29. Prantl, A., Imam, Sh., and Sarkar, V., Interfacing Chapel with traditional HPC programming languages, *Proc. 4th Conf. Partitioned Global Address Space (PGAS) Programming Model*, New York, 2010.

30. Ershov, A.P. and Il'in, V.P., Software packages: A technology for solving applied problems, *Preprint of Computing Center, Siberian Branch, USSR Acad. Sci.*, Novosibirsk, 1978, no. 121.

31. Butyugin, D.S. and Il'in, V.P., Solution of problems of harmonic electromagnetic field simulation in regularized and mixed formulations, *Russ. J. Numer. Anal. Math. Modell.*, 2014, vol. 29, no. 1, pp. 1–12.

Translated by Yu. Kornienko