ISSN 1019-3316, Herald of the Russian Academy of Sciences, 2016, Vol. 86, No. 2, pp. 118–126. © Pleiades Publishing, Ltd., 2016.
Original Russian Text © V.P. Il'in, 2016, published in Vestnik Rossiiskoi Akademii Nauk, 2016, Vol. 86, No. 4, pp. 316–326.

Review

Fundamental Issues of Mathematical Modeling

V. P. Il'in*

Institute of Computational Mathematics and Mathematical Geophysics, Russian Academy of Sciences, Siberian Branch, Novosibirsk, Russia

Novosibirsk State University, Novosibirsk, Russia

e-mail: ilin@sscc.ru

Received September 14, 2015

Abstract—Mathematical modeling of processes and phenomena is becoming increasingly more significant in various spheres of human activity, including the science and technology, production, and social spheres. Problems of its development are acquiring basic, applied, and technological aspects, the distinction and interlinks of which require philosophical and methodological conceptualization. The article published below highlights three problems: the automation of computations, their supercomputer representation, and the creation of large-scale modeling systems with an open-type integrated environment.

Keywords: mathematical modeling, computational methods and technologies, automation of algorithm construction, integrated software environment, interdisciplinary direct and inverse problems.

DOI: 10.1134/S101933161602009X

Since mathematical modeling is an extremely many-sided notion, it can be considered from various points of view. One of the possible trends in classification is by types of problems solved, which can relate to electromagnetism, thermal physics, elastoplasticity, fluid and gas dynamics, multiphase filtration, chemical kinetics, quantum phenomena, dynamic systems, etc. Noteworthy is the exceptional topicality of interdisciplinary direct and inverse problems. In addition, mathematical formulations can be represented differently—as differential and/or integral, classic or generalized, variational and mixed equations.

Another possible approach to this diversity is the prism of industry-specific applications: machine building, metallurgy, energy production, geophysics, weather and climate, ecology and disasters, biomedicine, materials science, and economics and society. RAS Presidium basic research program no. 1 on strategic trends in science (competition of 2014) "Basic Problems of Mathematical Modeling," available on the Internet, is largely based on this principle.

The next criterion for classification is computational methods and technologies used at different stages of computer experimentation: geometric and functional modeling, mesh generation, initial equation approximation, the solution of algebraic systems obtained, optimization methods for inverse problems, the processing and visualization of numerical results, and so on.

The propagation of modeling is prompted by external factors, such as the fantastic increase in the performance of multiprocessor computing systems (MCSs), the rapid development of methods of theoretical and applied mathematics, and the practical needs of the epoch of reindustrialization and breakthrough science-intensive technologies. All this increases the practical potential of modeling, turning it, on the one hand, into a third way of cognition along with theoretical and empirical (field) studies and, on the other, into a crucial condition for increasing labor productivity and the GDP.

A bottleneck is programmer labor productivity, the growth of which lags critically behind the growth rates of supercomputer capacities. Overcoming the long-overdue crisis of applied programming requires a new paradigm of its development. The existing long-term practice is the implementation of applied software packages (ASPs), either commercial or public, for concrete classes of problems. Examples of such products are ANSYS [1] and Nastran [2]. Developments of another type are program libraries that implement a totality of algorithms for a certain type of computational tasks. For instance, Netgen [3] is responsible for mesh generation, PETSc [4] is a suite of algebraic solvers, and so on. Still another variant, which is becoming increasingly popular nowadays, is instrumental computational systems OpenFOAM [5], DUNE (Distributed Unified Numerical Environment) [6], and the basic modeling system (BMS) [7].

* Valerii Pavlovich Il'in, Dr. Sci. (Phys.–Math.), is chief researcher at the Institute of Computational Mathematics and Mathematical Geophysics, RAS Siberian Branch, and a professor at Novosibirsk State University.

In the case of instrumental computational systems, we speak about the formation of an integrated open-source environment developed by a wide community. It supports all principal algorithmic stages of computational experimentation and makes it possible to develop on the fly ASPs, including closed ones, for concrete and diverse applications. Initial in this case is not problem-related but methodological and instrumental orientation. This approach makes it possible to overcome the fundamental contradiction between universality and effectiveness: if algorithms to solve a specific task (from an admissible wide class) are abundant, one can choose a sufficiently economical implementation.

The concept of open-source computational systems is based on the idea that, under the practically infinite manifold of problems with different individual characteristics, all of them are described by a finite set of mathematical models, the solving techniques of which can be "unscrambled"; the organization of work with such intellectual objects should be automated to an adequate extent. The creation of such a large software complex requires a long lifecycle, an extensible composition of models and algorithms, adaptation to the evolution of computer platforms, interaction with external products, flexible interfaces designed for users of different professional preparedness, and component architecture that would support the principles of modular (or assembling) programming. Actually, we are speaking about the creation of a global high-performance system of science-intensive technologies of a new generation, aimed at making cardinal changes in the demand for mathematical modeling and oriented at ensuring import-substituting software developments urgent for national security and highly competitive in the external market.

This article is dedicated to consideration of the above large-scale and inseparably interconnected issues: first, the automation of model and algorithm construction; second, the reflection of the algorithm structure on the architecture of present-day and future heterogeneous MCSs; and, third, the development of component architectures and technologies of creating very large unified complexes designed to allow applied software to have not only a high profitability and effectiveness but also flexible potentialities of wide-ranged use, comparable with the performance and interface qualities of operating systems or compilers.

AUTOMATION OF MODEL AND ALGORITHM CONSTRUCTION IS THE BASIS FOR APPLIED SOFTWARE INTELLECTUALIZATION

Irrespective of the subject orientation of the respective applied software, computational experimentation passes through similar technological stages.

Geometric and functional modeling. At the first stage, the user formulates a computation task, which can include the description of a complex geometric configuration of a domain consisting of subdomains with different material properties. If the geometric objects and operations have long been assimilated in numerous CAD products (CAD, CAE, CAM, PLM) and graphics systems, functional modeling requires operating with formalisms such as equations in subdomains, boundary conditions on border segments, various coefficients, and so on. In addition to source data, we should specify what we want to obtain and in what form. Methods to be applied or even detailed computational schemes, which unambiguously determine the process of mathematical modeling in a concrete environment, may also be prescribed. With the emphasis on the above aspects, we, in fact, come to the automation of model and algorithm construction.

The creation of flexible input and output interfaces for the end user remains a problem. Ideally, such an interface is represented by a natural-type programming language with the use of both text and graphic means. Since very different specialists—mathematicians, programmers, engineers, technicians, etc.—can use the same software, it is obviously necessary to form a rich variety of applied cognitive means. This, in turn, predetermines the task to organize a "factory" of numerous application-specific languages, which the literature calls Domain-Specific Languages (DSLs) [8].

Note that already the first stage of computer experimentation shows that the high demand for supercomputer technologies requires not only high-performance computations but also high-level artificial intelligence. Work [8] associates this point with transfer from "paleoinformatics" to "neoinformatics."

Problem discretization. Solution of nontrivial mathematical equations practically always begins with constructing a grid. To show the diversity of the questions that arise in this respect, it is enough to enumerate the most popular types of grids, such as adaptive; structured, unstructured, and quasi-structured; matching, nonmatching, and mortar; regular and irregular; static and dynamic; and so on.

The most effective approaches to discretization are associated with sufficiently complex discrete objects and their transformations, including sequences of hierarchical grids and their local zooms, decompositions of grid domains into subdomains, dynamic reconfiguration of grids, and a posteriori and/or a priori account for the properties of the desired solutions. Although there are quite a few indicators of the quality of grids, the determination of the optimal grid remains a very complicated problem, which practical developments do not even formulate. The most frequently used principle of choice is reducible to the empirical approach—the use of distribution densities of mesh

nodes proceeding from general qualitative considerations. Individual methodological recommendations relate to particular cases and are merely prove-the-rule exceptions. The global applied-software market simultaneously offers both very expensive and free mesh generators, which use a certain number of mesh data structures (MDSs), recognized by the computational community. Effective use of this colossal materialized intellectual potential appears a very topical task.

For a long time, basic and applied problems of mesh construction have been studied actively using various variational and engineering approaches. Since the problem of the optimality of meshes and their decomposition is extremely multifaceted, it has no sufficiently strict formulation thus far. We will return to this issue later; for the time being, let me note that today we can speak only about constructing "good" grids, i.e., those meeting approbated but rather numerous qualitative and quantitative criteria: triangles must not have angles that are too small, the volumes and areas of cells must not degenerate, etc. These criteria are either obtained from theoretical evaluations or derived purely empirically, which has even led to the opinion that one should construct "beautiful" grids, and, hence, it is quite natural to use visualization means to control mesh generators.

At the descriptive level, we can define the quality of mesh adaptivity, which is at least desirable, even if not mandatory: the nodes and edges of the sectionally smooth boundary of the computational domain must coincide with the nodes and edges of the grid, and the steps of the grid must be smaller where the derivatives of the desired solution are relatively large. The latter is calculated either on the basis of a priori theoretical evaluations or proceeding from a posteriori analysis of intermediate numerical results. We stress that the observance of these requirements can, as a rule, save by many times the total computational resources for obtaining a solution with the desired precision.

Discretization is a technological stage important for both resource intensity as a whole and computational resolution, which largely determines the success of modeling. This is especially true of problems with a complex spatiotemporal behavior of the solution, including actual situations with strongly multiscale characteristics. Therefore, mesh generation is a highly intelligent methodology; the lack of substantial theoretical and algorithmic results makes us orient not toward automatic but toward automated mesh construction accompanied by immediate visualization and the participation of an expert in controlling the computational process.

Approximation of equations. When, after the previous stages have been executed, an MDS is formed, which, together with the geometric and functional data structures (GDS and FDS), reflects all information about the initial problem at the discrete level, it is time to approximate it. The result is a system of finite-dimensional algebraic relations—an algebraic data structure (ADS) that can effectively use widespread matrix representations for sparse algebraic systems. As an example, let us give the Compressed Sparse Row (CSR) [9].

The operations performed in this case turn out to be the most science intensive and are represented by diverse theoretical approaches: finite-difference, finite-volume, and finite-element methods (FDM, FVM, and FEM) [10, 11]; different spectral algorithms; integral equation methods; and so on. The logical complexity of "approximators" particularly increases when methods of a high order of accuracy are used, especially on unstructured grids, formulas for which take several pages. It is this circumstance that hinders their wide occurrence despite their significant advantages.

The cardinal solution to this situation is the use of artificial intelligence potentialities, namely, the means of automating analytic symbolic transformations. In principle, such tools are present in large specialized systems of the Reduce or Maple type and are successfully used, for example, in the FEniCS and Helmholtz packages [12]. In the above cases, the problem is simplified by the FEM- and FVM-based unique polyelement technology of independent and easily parallelized computation of local matrices with subsequent assembly of the global matrix. Moreover, in proceeding from the additive methodology, it becomes possible to embrace the issues of accounting for boundary conditions of various types, autonomous approximations of standard differential Laplace operators, divergence, rotor, gradient, and so on, as well as their integral analogs in generalized variational formulations.

Solving algebraic problems. At this stage, various matrix–vector operations are performed that require the largest computer resources because the volume of arithmetic operations and necessary memory often grows nonlinearly here with growth in the number of degrees of freedom of the problem. The performed computations may imply implementing recurrent sequences, solving systems of algebraic equations (linear, SLAEs, and nonlinear, SNAEs), solving eigenvalue problems, and realizing optimization algorithms for mathematical programming. These tasks constitute vast fields of computational algebra, characterized by the colossal diversity of conceptual approaches, concrete versions of methods, and particular ways of their application. It is here where issues of parallelizing algorithms and reflecting them on MCS architectures, particularly on cluster systems containing heterogeneous nodes with classical and specialized processors, arise.

The international market offers a colossal amount of algebraic software, which is continuously updated

and expanded due to adaptive modifications for new computer platforms and architectures and the rapid development of new algorithms. Rapid growth and continuous updating create the problem of the coordinated reuse of the existing products. Note that there are serious achievements in this area: standard universal data structures and libraries with the basic set of matrix–vector operations (BLAS, SPARSE BLAS) [9].

The diversity of algebraic methods is due, first, to the different types of the participant matrices—Hermitian and anti-Hermitian, real and complex, symmetrical and antisymmetrical, degenerate and nondegenerate, positive-definite and indefinite, and so on. Matrices of all types are divided into dense and sparse, and approaches to their processing are significantly different. Moreover, the choice of optimal algorithms largely depends on structural properties of matrices (band, triangular, etc.), as well as on their dimensionality (the notion of "large" matrices continually changes depending on the capacity of the current generation of computers and, in the postpetaflops era, is related to 10^9–10^{12} orders of magnitude). Especially difficult are poorly worded problems with a strong instability of numerical solutions relative to inherent or computer round-off errors.

The most effective modern algorithms are characterized by a high logical complexity: multigrid approaches, domain decomposition methods, variable sequence optimization or matrix scaling techniques, and so on. We can state that the most resource-intensive algebraic methods also require the active use of artificial intelligence.

Optimization approaches to solving inverse problems. Although solutions of direct problems of mathematical modeling that require finding desired functions at all given coefficients of equations and initial and final boundary conditions can be of high computational complexity, they are usually only a part of the difficulties associated with the solution of an inverse problem. The latter is characterized by the fact that some of its initial data depend on unknown parameters, which should be found by minimizing the described objective functional under certain additional restrictions on the problem' properties. For example, when computing technical devices or instruments, the engineer usually aims not only at studying their properties but also at the computer-assisted design of optimal configurations that would ensure the required characteristics. In addition, practically always there are additional restrictions associated with the size, weight, or other functional conditions. Another characteristic example of an inverse problem is identification of the parameters of a mathematical model based on comparison of the estimated results with the data of field (indirect) measurements.

The main universal approaches to solving inverse problems are based on the use of conditional minimization methods, which imply a guided sequential search for a local or global minimum, the intermediate values of the objective functional being computed at each step, which is none other than the solution of a direct problem. Consequently, in the general case, the solution of an inverse problem requires repeated solutions of direct problems.

In recent decades, optimization methods have been developing actively, giving rise to new trends, such as algorithms of internal points, sequential quadratic programming, and confidence intervals [13]. Note, however, that the minimization of functionals with complex geometric characteristics, especially of the ravine type, is something at the interface of science and art; this is why a fully automated computational process is possible only in the simplest situations. In fact, even in this case, highly intelligent technologies are necessary, implying step-by-step implementation of the entire problem in dialogue with the user, who, proceeding from his experience, should control the behavior of sequential approximations and govern the parameters of algorithms to achieve the ultimate goal as soon as possible.

The postprocessing and visualization of results. Computational process control and decision-making tools. The results of algebraic computations lack any physical meaning and obviousness, primarily owing to their large volumes. For example, FEM makes it possible to obtain coefficients of the expansion of required solutions by the basic functions used in grid cells, while the user needs a compact and illustrative picture of multidimensional vector fields. Hence, applied software should have a developed set of instruments to form typical representations, such as isosurfaces, force lines, cross sections, various graphs, and so on. This is the first requirement. The second one is associated with the fact that one cannot foresee everything, and an intelligent modeling system should contain means of automating the programming of various possible characteristics of final data. Finally, the third factor is the possible diversity of the professions of end users, who want to receive a comfortable representation of the results of using the computer, which determines its production effect.

Importantly, even ideal applied software does not eliminate the fact that computer modeling of complicated processes or phenomena is a multifold creative activity. For example, to study some applications systemically, you should first make sure that the models and methods applied meet the specifications; for this, it is necessary to perform test computations first and then to analyze whether the obtained data are adequate. Then it comes to the studies proper, which can be a large-scale machine experiment preceded by planning and method selection procedures. The latter

is unattainable without providing for flexible possibilities to compile computational schemes, which implies the creation of respective languages (declarative or imperative) to control computational processes. Finally, since modeling is not an end in itself but a tool for cognitive or production activity, then, to ensure decision making in line with computational results, the applied software should contain either some cognitive principles or means of connecting to CAD infrastructures, or technologies of supporting and optimizing the operation regimes of concrete processes. However, these issues are beyond mathematical modeling proper.

PARALLELIZING ALGORITHMS AND REFLECTING THEM ON MCS ARCHITECTURE

One of the main trends in the development of computational sciences and technologies is convergence of algorithmic structures and computer architectures. Since the latter of the above sides of the process turns out to be outside the topic considered in this article, let us limit ourselves to analysis of cluster systems with heterogeneous nodes that contain classical (central) and graphic processors with numerous compute cores, which implement algorithms using hybrid programming with the organization of Message–Passage–Interface (MPI) processes in distributed memory and multithreaded operations in shared memory.

Some general issues of parallelization. A universal requirement on applied software is the absence of software restrictions on the number of degrees of freedom (d.o.f.) of the problem under solution and the quantity of the processors and/or cores used. Note also the important algorithm parallelization characteristics such as weak and strong scaling. Weak scaling means that computation time remains practically the same under the simultaneous increase in d.o.f. and the number of computing devices, while strong scaling is a proportional decrease in the time of a fixed problem with the growth of the number of computers.

Ideally, a solution to the problem of the automation and optimization of algorithm parallelization should be sought through the simulation modeling of a computer system as a whole. However, this is too complicated; this is why one has to employ semiempirical techniques or the simplest models of computer calculations. Examples of parallelization characteristics are two values: the coefficients of computational speedup and processor utilization efficiency:

$$S_p = T_1/T_p, \quad E_p = S_p/P,$$

where T_p is the time of performing a task or an algorithm on processors (p). The ideal situation is when the value of S_p is directly proportional and $E_p = 1$ is equal to unity; in practice, however, we often have to content ourselves with efficiency factors of several percent.

The development of supercomputer technologies proceeds in two main directions: high-performance computing (HPC) and working with Big Data; note that the convergence of these two trends (intensive data computing) has been observable of late. Overall, the evolution of MCS generations and extremum modeling problems is accompanied by similar growths of RAM speed and capacity (the number of tera- or petaflops is quite similar to the number of tera- or petabytes of the computer).

The main goal in programming parallel algorithms is to minimize information exchange because total problem time T equals the sum of two addends:

$$T_a = N_a\tau_a, \quad T_c = M(\tau_0 + N_c\tau_c),$$

where τ_a and τ_c are the average times of one arithmetic operation and one number transfer, N_a is the number of arithmetic operations, M is the number of memory accesses, τ_0 is exchange operation delay (setting) time, and N_c is the average volume of one array under transfer. We should bear in mind the characteristic relation $\tau_0 \gg \tau_c \gg \tau_a$. The requirement on decreasing data transfer is explained not only by the necessity to increase the speed but also by the energy consumption of communications.

The above makes it clear that the notion of the quality of algorithms changes for large tasks: out of the two methods compared, the best is not the one that has fewer computations but the one that is realized quicker on MCSs of the type under consideration. In other words, there appears a new concept of computation optimization based on the search for approaches that significantly decrease the volume of data transfer between the processors, even if they increase the number of arithmetic operations.

The term *large problem* requires a formal definition. In this case, we mean a problem that requires much computing time. In a sense, this notion is an invariant relative to the generation of the computer. In addition, it is necessary to concretize the category "extreme modeling," i.e., solving large (or superlarge) problems on supercomputers, such as, for example, MCSs that, in terms of power-related indicators, are ten times inferior to the last computer from the current world TOP-500 list.

Speaking about intensive computations, one cannot but mention the basic problem of stability to round-off and inherent errors. Let me limit myself to stating that solving very "bad" or ill-defined (so-called stiff, ill-posed) problems requires specific conditions for result accuracy control and, quite often, the search for special algorithms. Except for such exceptional situations, the estimated errors are determined by the length of the mantissa in the floating-point number

representation. At present, for moderately and highly complex problems, "double-precision" arithmetic operations are universally used, which corresponds to a 64-bit machine representation of real numbers. On the one hand, this significantly simplifies algorithm optimization; on the other, optimization-related issues are simply put outside the brackets. The ideal solution would be the use of variable word length; however, this strategy is viewed today as a distant prospect.

The last point of interest is the necessity to overcome the mental complex when we have no supercomputer "within reach." With modern cloud technologies, it is sufficient to have Internet access to the computing center for collective users (CCCU, or Data Center). Of course, to intellectualize the user interface, the workstation should be equipped with specialized means; however, this is outside the scope of this article.

Characteristic features of the parallelization of technological studies. Parallelization tactics at each computation stage are determined by the volume of data and the number of operations. The stage of geometric and functional modeling, substantial in intellectual loads and crucial for the user input interface, deals with macroobjects, which should not be too many (tens, hundreds, or, at worst, thousands). Hence, it seems desirable to do without exchanges, copying the computations in all MPI processes and storing in them the geometric and functional data structures obtained.

Mesh generation may formally be represented as data transformation: GDS + FDS –> MDS; note that the mesh data structure for the entire computational space can be of great volume. Due to this, it is natural to create an MDS by each MPI process for "its" mesh subdomain (with a certain overlap). The formation of distributed data at the initial stage is reasonable, the more so that the decomposition of domains is the main instrument of parallelization. However, since the estimated mesh domain should also be identified as an integral object, all its nodes and other elementary objects (edges, faces, cells) should be numbered twice—locally by subdomain and globally. Decomposition problems can employ two tactics: subdomain construction, which precedes mesh generation (for example, it is natural to separate media with contrastingly different material properties), or the direct formation of mesh subdomains. We should also bear in mind that many effective algorithms are based on special rearrangements of components (one may speak about this task in terms of graphs as well), and all the respective procedures should be accessible for all MPI processes, or subdomains, which, overall, will substantially reduce data exchange. The popular software packages METIS and PARMETIS are effective rearrangement instruments.

The most computationally complex are moving boundary problems, because they imply that adaptive grids are dynamically reconfigurable as well. Many economical methods are based on local zooms and multigrid approaches, whose instrumental support should also be distributed.

Upon obtaining distributed data arrays, reflected in MDS, GDS, and FDS, one can in parallel approximate the initial problem. For this purpose, FEM and FVM have a unique technology of computing local matrices and assembling the global matrix. Since the "approximator" works in parallel by subdomain, or MPI processes, with already distributed necessary data, the obtained matrix–vector structures must be in their subdomain; hence, this stage can be realized perfectly in the absence of exchanges. Principal operations performed by mesh cells independently of one another can be parallelized effectively using multithreaded computing. In nonstationary problems, as well as in nonlinear or optimization computations, approximations are repeated; however, from the point of view of adaptation to computing devices, this usually changes nothing.

The most important intermediate element in solving algebraic problems is linear systems; this is why we will focus on them. Special attention should be paid to very large SLAEs with sparse matrices, which emerge after the FEM- or FVM-assisted approximation of differential or respective variational multidimensional problems on unstructured grids. From the point of view of the classification of algorithms, the SLAEs under solution can be divided into two major classes—special and general ones. For the former, which comprises systems that emerge in boundary value problems with separable variables, there are superfast direct and/or iterative problem solvers, like the fast Fourier transform or alternating direction implicit methods with optimal sets of iterative parameters [10], which have been in demand over the last decade due to actual Lyapunov and Sylvester matrix equations.

Direct methods for large sparse SLAEs of the general type are improving actively; however, even in the most advanced versions of the popular Pardiso [9] and MUMPS programs, their applicability is limited, mainly because of their requirements on the RAM volume. The main tool for a parallel highly productive solution of SLAEs of this type is iterative additive domain decomposition methods (DDMs), which are covered in the multitudinous special literature (for example, see the review in [14]) and have been discussed at 23 major international conferences. The essence of these decomposition methods is dividing a computational mesh domain into subdomains with parametrized intersections (in the particular case, without intersections), at the internal boundaries of which certain interface boundary conditions are set to determine informational interrelations between

neighboring subdomains. In the simplest case, iterations are formed according to the Jacobi decomposition method, which implies solving auxiliary SLAEs in subdomains simultaneously with data exchange between them. To accelerate this process, optimal algorithms in Krylov subspaces are primarily used. For further increase in speed, various two- or multilevel approaches are employed, such as deflation, aggregation, coarse-mesh correction, low-rank matrix approximations, etc.

To attain scalable parallelization, hybrid programming technologies are used: MPI processes are formed over the memory distributed by computational nodes, one per subdomain, inside which multithreaded computations are performed using OpneMP in common memory. Note that substantial acceleration is achievable if interprocessor exchanges are matched with synchronous performance of arithmetic operations in subdomains. A separate problem is how to use effectively universal graphics accelerator cards with a great number of computer cores but relatively slow communications (General Purpose Graphic Processor Units, GPGPUs), as well as advanced field programmable logic devices (FPLDs) or free programmable gate arrays (FPGAs).

The adaptation of modern decomposition methods to the existing computer platforms is, in terms of philosophy and methodology, a problem of reflecting algorithms to the MCS architecture. This basic (in terms of significance) scientific trend is largely experimental, and only numerous comparisons of the measurements of real productivity can be the foundation for elaborating practical recommendations on solving classes of problems.

The postprocessing and visualization of computational results is a sphere most favorable for parallelization. With its seeming mathematical simplicity, this technological stage is key to the success of large modeling projects. High-quality color graphics, especially with dynamic scenarios and repeated control of intermediate data, requires significant computer resources and, in a large-scale computational experiment, can take the lion's share of machine time. Since one of the main requirements on the quality of visualization is a high speed of image generation, a natural technical solution is the use of high-speed GPU graphic processors. An important feature of visualization is that the resultant multidimensional vector fields, which should be graphically presented to the user, are distributed over hierarchical memory units of different processors. Another circumstance is related to the presence of a large number of professional graphic products (Visual Studio, Open GL, and so on), and one of the main problems for developers of a modeling system is their effective reuse.

From the point of view of large-scale parallelization, optimization methods and computational experimentation control are a superstructure over resource-intensive computing stages, and we can expect no special problems here, although the decisions made at the upper block level play a significant role in reaching the final high performance.

THE COMPONENT ARCHITECTURE OF THE INTEGRATED ENVIRONMENT OF MATHEMATICAL MODELING

The roadmap of Jack Dongarra's international committee (International Exascale Software Project, IESP) emphasizes that the rise of exaflops supercomputers with several million computer cores by 2020 requires the creation of a new programming paradigm and large-scale developments, which are possible only under wide and well-coordinated international cooperation [15]. In the sphere of mathematical modeling, this challenge requires forming and implementing principles of constructing applied software of a new generation, namely, creating an integrated instrumental environment that would support all stages of science-intensive experimentation and serve as the basis for operational developments of software packages for concrete applications and users. This method–problem–oriented complex should constitute a new generation of applied software, which is being created by numerous groups of developers on the basis of coordinated technologies and data structures. The large-block configuration of this globalized project, which can be called the basic modeling system (BMS), is determined by the computational stages considered above from the functional point of view.

The BMS concept and principles of constructing major instrumental blocks are considered in works [7, 16–20]. This project is based on the following provisions:

• the BMS is created as a rich and even abundant, but by no means minimal set of computational methods to solve a wide range of applied problems, including inverse and interdisciplinary ones, with possibilities of a flexible choice and optimization of algorithms proceeding from the properties of a concrete problem to be solved;

• the composition of algorithms and mathematical models is extensible and operationally replenishable, among other things, by using external software products;

• program implementations of modules are adapted to the evolution of computer architectures and platforms, opening up prospects for a long lifecycle of the BMS;

• ensuring internal and external programming interfaces, as well as interaction with users, is based on the multiple representation of data structures with the active use of their conversion and specialized languages;

• algorithms are realized as oriented to mass scalable parallelization, without software restrictions on the number of degrees of freedom of a problem and the number of processors and/or cores used and on typical technologies of large problem solutions—cloud computations on CCCU supercomputers.

As for the BMS architecture, each of its structural, or stage, blocks is actually an extensible library of algorithms and programs, i.e., an integrated and sufficiently autonomous environment for a respective method–environmental niche. Let me stress that the blocks are constructed on the principles of the multiplicity of both data structures and algorithms for solving particular subtasks. For example, this multiplicity can include implementation multiversionality in different programming languages or for different computing devices (central of graphics processors). The functionally oriented blocks under consideration can be elaborated practically independently from one another (and even be operated as independent products in other production conditions), and interaction between them is determined only by interface data structures (GDS, MDS, ADS, and so on).

The component technologies that have widely been used for several decades, such as Common Object Request Broker Architecture (CORBA) [21] and COM/DCOM (Distributed Component Object Management) [22], are in a sense the development of object-oriented programming; however, they do not realize the duly high performance of science-intensive computations with scalable parallelism, complex types of data, and multilingual program modules. To overcome these limitations, the Common Component Architecture (CCA) Forum [23] was organized in 1997, which set the task to determine the main standards, instruments, and technologies. The US Department of Energy established the Center for Component Technology for Terascale Simulation Software (CCTTSS), which, jointly with leading national laboratories, created a number of instrumental means within the CCA specifications [24]. Among such developments, noteworthy are the scientific interface definition language (SIDL) [25], which is the generalization of the IDL language from the CORBA project to support multilingualism, including F77, F90/95, C, C++, and Python, implemented in the Babel Team [26], as well as the instrumental complex CCAFFEINE [26] for conducting technological operations in hierarchical distributed memory. Today there are already a number of publications on the results of the successful use of component technologies for serious applications. An example is the SPARSKIT software package [27] upgrades within the CCA methodology to solve problems of linear algebra, as well as interface standards to solve equations, developed in the SANDIA (United States) national laboratory [28].

Overall, the ideas of computer architectures echo other programming technologies—modular, assembling, and fragmentary ones. In recent years, the trend of service-oriented architectures (SOA) has appeared, which also implies a cardinal increase in the automation (and, consequently, productivity) of the labor of programmers. This is largely associated with the reduction of the times of the labor-intensive software-debugging process, which includes repeated passes of test routines and the search for inevitable technical errors. The developer must code the algorithmic module only according to the specified requirements on the representation of input and output information, while the programming system should ensure its correct execution within the entire complex irrespective of the operation system and computing device that will launch it (platform independence).

The property of multilingualism here is no less important for two reasons: the first is provision for the possibility to reuse third-party modules in the BMS; the second is determined by the tendency to create new languages customized for writing algorithms of certain classes quickly. Such natural or functional languages with a wide use of not only imperative (command), but also declarative, styles significantly increase the intellectuality of human–computer interaction and are ultimately aimed at increasing the effectiveness of modeling. The automation of algorithm parallelization at the language level remains an attractive prospect, but numerous attempts have not led to the development of a commercial product thus far.

* * *

Because of the increased complexity of tasks in basic research, as well as in breakthrough industrial and social technologies, the globalization of mathematical modeling on supercomputers has become a major requirement of scientific–technological advance. However, we should, first, bear in mind that its wide-scale introduction requires the solution of a great number of production, organizational, and personnel issues [29] and, second, distinguish between modeling problems proper and those relating to disciplines and industries in which it is used. We can draw the following analogy: mathematicians study mathematical objects; theoretical physicists actively use various formulas and methods for their purposes but do not prove theorems. At present, instead of the qualitative analysis of the properties of differential or integral equations, on which physicists often spend their time (although, in reality, this is possible only in the simplest cases), they can take a “smart” applied software product and obtain quickly and graphically the necessary results, investigating all possible variants on the fly. The same is also true of engineers engaged in optimizing a device under design. However, even if there

exists a perfect instrument of modeling, the art of its effective use is the problem of users in concrete subject areas, which also requires close attention on the part of researchers.

ACKNOWLEDGMENTS

This work was supported by the Russian Science Foundation, grant no. 15-11-10024.

REFERENCES

1. ANSYS—Simulation Driven Product Development. http://www.ansys.com. Cited January 10, 2015.
2. MSC—Industry Leading Multidisciplinary FEA. http://www.mscsoftware.com/product/msc-nastran. Cited February 11, 2015.
3. J. Schoberl, "Netgen - àn advancing front 2D/3D-mesh generator based on abstract rules," Comput. Visual. Sci. **1**, 41 (1997).
4. PETSc. http://www.mcs.anl.gov/petsc/ Cited February 5, 2015.
5. Open FOAM—The Open Source Computational Fluid Dynamics (CFD) Toolbox. http://www.openfoam.com. Cited April 1, 2015.
6. DUNE Numerics. Distributed Unified Numerical Environment. http://www.dune-project.org. Cited January 15, 2015.
7. V. P. Il'in and I. N. Skopin, "Computational programming technologies," Program. Comput. Software **37** (4), 210 (2011).
8. A. Kleppe, *Software Language Engineering: Creating Domain-Specific Language Using Metamodels* (Addison-Wesley, New York, 2008).
9. Intel ® Math Kernel Library (Intel® MKL). http://software.intel.com/en-us/intel-mkl. Cited March 2, 2015.
10. V. P. Il'in, *Methods of Finite Differences and Finite Volumes for Elliptical Equations* (Izd. IVMiMG SO RAN, Novosibirsk, 2001) [in Russian].
11. V. P. Il'in, *Methods and Technologies of Finite Elements* (Izd. IVMiMG SO RAN, Novosibirsk, 2007) [in Russian].
12. D. S. Butyugin and V. P. Il'in, "Solution of problems of harmonic electromagnetic field simulation in regularized and mixed formulations," RJNAMM **29**, 1 (2014).
13. V. P. Il'in, "Numeric solving of direct and inverse problems of electromagnetic prospecting," Sibir. Zh. Vychisl. Mat. **6**, 381 (2003).
14. V. P. Il'in, "Parallel processes at the stages of petaflops modeling," Vychisl. Metody Program. **12**, 93 (2011).
15. IESP: International Exascale Software Project. http://www.exascale.org/iesp. Cited March 02, 2015.
16. L. A. Golubeva, V. P. Il'in, and A. N. Kozyrev, "Programming technologies in geometric aspects of mathematical modeling," Vestn. NGU. Ser. Inform. Tekhnol. **10**, 25 (2012).
17. V. P. Il'in, "DELAUNAY: A technological environment for mesh generation," Sib. Zh. Ind. Mat. **16**, 83 (2013).
18. D. S. Butyugin and V. P. Il'in, "Chebyshev: Principles of automation of algorithm construction in an integrated environment for mesh approximations of initial boundary value problems," in *Proc. Int. Conf. PAVT 2014* (Izd. YuUrGU, Chelyabinsk, 2014), pp. 42–50 [in Russian].
19. D. S. Butyugin, Ya. L. Gur'eva, V. P. Il'in, et al., "Functionality and technologies of algebraic solvers in Krylov's library," Vestn. YuUrGU. Ser. Vychisl. Mat. Inform. **2**, 92 (2013).
20. V. P. Il'in, "Component technologies of high-performance mathematical modeling," in *Proc. Int. Conf. PAVT-2015* (UFU, IMM UrO RAN, Yekaterinburg, 2015), pp. 166–171 [in Russian].
21. Object Management Group. "CORBA Components." http://www.omg.org. Cited February 10, 2015.
22. J. Maloney, *Distributed COM Application Development Using Visual C++* (Prentice Hall, New York, 1999).
23. http://www.CCA-forum.org. Cited February 10, 2015.
24. http://www.cca-forum.org/ccttss. Cited March 10, 2015.
25. S. Kohn, G. Kumfert, J. Painter, and C. Ribben, Divorcing language dependencies from a scientific software library. https://computation.llnl.gov/casc/components/docs/2001-siam-pp.pdf. Cited March 7. 2016.
26. Babel Team. The DOE Babel Project. http://www.llnl.gov/case/components/babel. Cited April 10, 2015.
27. B. Allan, R. Armstrong, A. Wolfe, et al., "The CCA core specification in a distributed memory SPMD framework," Concurrency Computation. Practice and Experience **14**, 323 (2002).
28. J. Jones, M. Sosonkina, and Y. Saad, "Component-based iterative methods for sparse linear systems," Concurrency and Computation. Practice and Experience **19**, 625 (2007).
29. V. P. Il'in, "Computational mathematics and informatics: Global challenges and Russia's roadmap," Herald Russ. Acad. Sci. **85** (1), 8 (2015).

Translated by B. Alekseev

SPELL: 1. replenishable