Parallel Numerical Algorithms

On the Parallel Strategies in Mathematical Modeling

Valery Il'in(✉)

Institute of Computational Mathematics and Mathematical Geophysics RAS, Novosibirsk State University, 6 Pr. Lavrentieva, 630090 Novosibirsk, Russia
ilin@sscc.ru

Abstract. The article considers parallel strategies and tactics at different stages of mathematical modeling. These technological steps include geometrical and functional modeling, discretization and approximation, algebraic solvers and optimization methods for inverse problems, post-processing and visualization of numerical results, as well as decision-making systems. Scalable parallelism can be provided by combined application of MPI tools, multi-thread computing, vectorization, and the use of graphics accelerators. The general method to achieve high-performance computing consists in minimizing data communications, which are the most time and energy consuming. The construction of efficient parallel algorithms and code optimization is based on various approaches at different levels of computational schemes. The implementation of the biggest interdisciplinary direct and inverse problems in cloud computing technologies is considered. The corresponding applied software with a long life cycle is represented as integrated environment oriented to large groups of end users.

Keywords: Scalable parallelism · Domain decomposition · Runtime · Communications · Multi-thread computing · Vectorization · Exchange buffers · Hierarchical memory · Speedup · Accelerators

1 Introduction

The idea of parallelization is very old, and consists in the simultaneous operation of different hardware units. Modern heterogeneous supercomputer multiprocessor systems (MPS) have a rich architecture: a large net of nodes with distributed memory, sets of multi-core CPUs with shared hierarchical memory and very fast registers, and several graphics accelerators (at each node) of GPGPU or Intel Phi type, with a complicated internal structure. We can, accordingly, consider four-level hybrid programming tools: Message Passage Interface (MPI system), multi-thread computing (OpenMP), CUDA system, and vectorization of machine operations by applying AVX instructions inside the CPU or the Phi

The work was supported by the RFBR grant N 16-29-15122 and the RSF grant N 15-11-10024.

L. Sokolinsky and M. Zymbler (Eds.): PCT 2017, CCIS 753, pp. 73–85, 2017.
DOI: 10.1007/978-3-319-67035-5_6

unit. Moreover, programmers can use different hints for code optimization, taking into account detailed peculiarities of the memory access. We should emphasize that the evolution of computer platforms is dramatically fast, and applied software must be flexibly adapted to hardware changes in order to provide a long numerical life cycle of the environment.

Scalable parallelism is a challenging problem when solving interdisciplinary direct and inverse super-tasks concerned with mathematical modeling (MM), which now constitute the main tool for obtaining new fundamental knowledge and optimizing industrial production. One of the main trends of development of computational and informational technologies (CIT) to study various processes and phenomena consists in the convergence of MM approaches and CAD-CAE-CAM applications. Other important feature of the current situation is associated with a rapid upsurge of new results in theoretical and computational mathematics. It is well-known that the most "clever" and efficient algorithms are difficult to parallelize. Thus, the most urgent issue concerning MM support is to strike the right balance between the mathematical efficiency of an algorithm and the computer performance of its program implementation.

The bottleneck is programmer's labor productivity, which lags far behind the growth rates of supercomputer capacity. Overcoming the world-wide crisis requires a new paradigm of development. The existing long-term practice is the implementation of applied software packages (ASPs), either commercial or publicaccess, for concrete classes of problems. Examples of such products are ANSYS [1] and FeniCS [2]. Developments of other types 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. Another versions that are becoming increasingly popular nowadays are instrumental computational systems OpenFOAM [5], DUNE (Distributed Unified Numerical Environment) [6] and Basic System of Modeling (BSM) [7]. Some general issues that arise when creating a program environment for mathematical modeling are considered in [8]. It is worth mentioning an interesting project devoted to algorithms and their parallel implementations: the Open Encyclopedia of Properties of Algorithms [9].

This paper is organized as follows. Section 2 contains the algorithmic description of the main technological stages of mathematical modeling. In Sect. 3, we discuss specific features of parallel tactics at different steps of a large-scale numerical experiment. In the Conclusions, we make some remarks on parallelization strategies for cloud computing and Data Center frameworks.

2 Technological Stages of Large-Scale Numerical Experiments

Regardless of the subject orientation of applied software, a computational experiment goes through similar technological stages. We can implement these steps almost independently if we define the internal interfaces correctly, in accordance

with Virt rule: "Program = Algorithms + Data Structure". Some performance and intellectuality issues of supercomputer modeling are discussed in [10].

Geometric and functional modeling. At the first stage, the user formulates a computational task, which may include a description of a complex geometric configuration consisting of subdomains with different material properties. While geometric objects and operations have long been assimilated in numerous CAD products (CAE, CAM, PLM) and graphics systems, functional modeling requires operating with formalisms such as equations in subdomains, boundary conditions on border segments, various coefficients, etc.

The formal description of an initial boundary value problem for partial differential equations (PDEs) can be presented, for example, as follows:

$$\begin{aligned} &L\boldsymbol{u} = f(\boldsymbol{x}, t), \quad \boldsymbol{x} \in \bar{\Omega}, \ 0 < t \le T < \infty, \\ &l\boldsymbol{u} = \boldsymbol{g}(\boldsymbol{x}, t), \quad \boldsymbol{x} \in \Gamma, \ \boldsymbol{u}(\boldsymbol{x}, 0) = \boldsymbol{u}^0(\boldsymbol{x}), \\ &L = A\frac{\partial}{\partial t} + \nabla B \nabla + C\nabla + D, \ \ \Gamma = \Gamma_D \cup \Gamma_N, \\ &\boldsymbol{u}\big|_{\Gamma_D} = \boldsymbol{g}_D, \ \ (D_N \boldsymbol{u} + A_N \nabla_n \boldsymbol{u})\big|_{\Gamma_N} = \boldsymbol{g}_N. \end{aligned} \tag{1}$$

Here $\boldsymbol{x}$ and t are spatial and time variables, L and ℓ are differential operators, $\boldsymbol{u}$ is the solution sought (in general, a vector), A, B, C, D are some matrices, and Γ_D, Γ_N are border segments with different types of boundary conditions.

In addition to input data, we should specify what we want to obtain and in which 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. Emphasizing the above aspects, we have come, in fact, to the automation of the model and algorithm construction. Some questions on these topics, including geometric and functional data structures (GDS and FDS), are discussed in [11].

Problem discretization. The solution of non-trivial mathematical equations essentially always begins with constructing a grid. To show the diversity of questions that arise in this respect, it suffices to mention 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. Modern real super-tasks require a quite large number of nodes (10^9 and more). Important questions on the performance of this stage, as well as a review of algorithms and numerical software are presented in [12].

The most effective approaches to discretization are associated with sufficiently complex discrete objects and their transformations, including sequences of hierarchical grids and their local refinement, decomposition of grid domains into subdomains, dynamic reconfiguration of grids, and an *a posteriori* and/or *a priori* account of 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 studies do not even formulate. The most frequently used principle of choice can be reduced to an empirical approach: the use of distribution densities of mesh nodes according to

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

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

Approximation of equations. When the previous stages have been executed and an MDS has been formed, which, together with the geometric and functional data structures (GDS and FDS), reflects the whole information about the initial problem at a discrete level, its approximation becomes possible. The result is a system of finite-dimensional algebraic relations, i.e. an algebraic data structure (ADS) that can effectively use widespread matrix representations for sparse algebraic systems. As an example, let us mention the Compressed Sparse Row format (CSR).

The operations performed in this case are the most science-based and are represented by diverse theoretical approaches: finite-difference, finite-volume and finite-element methods (FDM, FVM and FEM); different spectral algorithms; integral equation methods, etc. The logical complexity of the "approximators" particularly increases when methods of a high order of accuracy are used, especially on unstructured grids, formulas for which would eventually extend over several pages. This circumstance hinders their wide dissemination despite their significant advantages. A 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 types, and are successfully used, for example, in the FEniCS package [2]. In the above cases, the problem is simplified by the FEM- and FVM-based unique element-by-element technology of independent and easily parallelized computation of local matrices with the subsequent assembly of a global matrix. Some general questions of approximation techniques are described in [13].

Solving algebraic problems. At this stage, various matrix-vector operations are performed that require the largest computer resources, since the volume of both arithmetic operations and necessary memory often grows nonlinearly as the number of degrees of freedom (d.o.f.) of the problem grows. The performed computations may require implementing recurrent sequences, solving systems of algebraic equations (linear, SLAEs, and nonlinear, SNAEs), solving eigenvalue problems,

and optimizing algorithms for mathematical programming. These tasks constitute vast fields of computational algebra, characterized by a colossal diversity of conceptual approaches, concrete versions of methods, and particular ways of their application. It is here where the issues of parallelization of algorithms and their implementation on MPS architectures, particularly on cluster systems containing heterogeneous nodes with classical and specialized processors, arise.

The international market offers a big amount of algebraic software, which is continuously updated and expanded, owing to adaptive modifications for new computer platforms and architectures, and the rapid development of new algorithms. The rapid growth and regular updating create the problem of coordinated re-use of existing products. Let us 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) [14].

The diversity of algebraic methods is associated, first of all, with a variety of types of considered matrices: Hermitian and non-Hermitian, real and complex, symmetrical and non-symmetrical, degenerate and non-degenerate, 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 dimension (the notion of "large" matrices continuously changes depending on the capacity of the current generation of computers, being in the post-petaflops era 10^9 to 10^{12} orders of magnitude). Ill-posed problems with a strong instability of numerical solutions relative to inherent or computer rounding errors are particularly complicated.

The most efficient modern algorithms are characterized by high logical complexity: algebraic multigrid approaches, domain decomposition methods, variable ordering optimization, matrix scaling techniques, and so on. We can affirm that the most resource-intensive algebraic methods also require an active use of artificial intelligence. The conception of an integrated numerical environment for computational algebra is presented in [15].

Optimization approaches to solving inverse problems. The solution of direct problems of mathematical modeling (1), which require to find the desired functions, given the coefficients of the equations and the initial and boundary conditions, may have a high computational complexity. But this is 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-aided design of optimal configurations that would ensure the required characteristics. In addition, almost always there are additional restrictions associated with the size, weight or other functional conditions. Another characteristic example of an inverse problem is the identification of the parameters of a mathematical model based on comparison of estimated results with data from natural measurements.

The optimization statement of an inverse problem can be written in the form

$$\begin{aligned} &\Phi_0(\boldsymbol{u}(\boldsymbol{x},t,\boldsymbol{p}_{opt})) = \min_{\boldsymbol{p}} \Phi_0(\boldsymbol{u}(\boldsymbol{x},t,\boldsymbol{p})), \;\; \boldsymbol{p} = \{p_k\}, \\ &L\boldsymbol{u}(\boldsymbol{p}) = \boldsymbol{f}, \;\; p_k^{min} \le p_k \le p_k^{max}, \;\; k = 1, ..., m_1, \\ &\Phi_l(\bar{u}(\boldsymbol{x},t,\boldsymbol{p})) \le \delta_l, \;\; l = 1, ..., m_2, \end{aligned} \tag{2}$$

where Φ_0 is the goal functional, $\boldsymbol{p}$ is an unknown vector parameter, $p_k^{\min}$ and $p_k^{\max}$ define linear constraints, Φ_ℓ and δ_ℓ are non-linear constraints, and $L\,u(\boldsymbol{p}) = \boldsymbol{f}$ denotes a constitutive equation that is defined, in fact, by the whole direct problem (1).

The main universal approaches to solving inverse problems rely on the use of constrained minimization methods, which imply a directed sequential search for a local or a global minimum and the intermediate values of the objective functional being computed at each step, which is nothing but 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 actively developed, giving rise to new trends, such as algorithms of interior points, sequential quadratic programming and trust regions. Note, however, that the minimization of functionals with complex geometric characteristics, especially those of the ravine type, is something at the interface between science and art. That 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, entailing a step-by-step implementation of the entire problem in dialogue with the user, who, based upon his experience, should control the behavior of sequential approximations and govern the parameters of the algorithms to achieve the ultimate goal as soon as possible.

Post-processing 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, the FEM makes it possible to obtain the coefficients of the expansion of the required solutions with respect to the basic functions used in grid cells, whereas the user needs a compact and illustrative picture of multidimensional vector fields. Hence the reason why applied software should have a developed set of instruments to construct 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 the means for automating the programming of various possible characteristics of final data. Finally, the third factor is that end users may come from different professions, and all of them want to obtain a comfortable representation of the results of using a computer, determining its production effect.

It is important that even ideal applied software does not preclude the fact that computer-aided modeling of complicated processes or phenomena is a multifold creative activity. For example, to study some applications systemically,

one should first make sure that the models and methods applied meet the specifications; for this, it is necessary, first, to perform test computations and then to analyze whether the data obtained are adequate. Then it appears possible to start the study itself, which can be a large-scale machine experiment preceded by planning and method-selection procedures. The latter are unattainable without providing for the flexible to compile computational schemes, which implies the creation of the corresponding languages (declarative or imperative) to control the computational processes. Finally, modeling is not an end in itself but a tool for cognitive or production activities, therefore, to ensure the adoption of a decision with computational results, applied software should contain either some cognitive principles or means for connecting to CAD infrastructures or technologies that support and optimize the operation regimes of concrete processes. However, these issues are beyond the frame of mathematical modeling.

3 Scalable Parallelism: Problems and Solutions

Modern mathematical modeling offers a huge set of real applications, models and numerical methods, as well as an essential diversity of hardware and computing platforms. This provides a great possibility to choose tactics and strategies for the optimization of computational processes.

Some general issues of parallelization. A universal requirement on applied software is the absence of software restrictions on both the number of d.o.f. of the problem to be solved and the quantity of processors and/or cores used. At the same time, it should be recalled that there are important algorithm parallelization characteristics such as weak and strong scaling. Weak scaling means that computation time remains practically the same as the number of d.o.f. and the number of computing devices grow. Strong scaling means a proportional time decrease for a fixed problem as the number of computers grows.

Ideally, a solution to the problem of automation and optimization of algorithm parallelization should be sought through simulation of a computer system as a whole. However, this is too complicated: that is why one has to employ semi-empirical techniques or the simplest models of computer calculations. Two values can be mentioned as examples of parallelization characteristics, namely the coefficients of computational speedup and the processor utilization efficiency:

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

where T_p is the time required for the execution of a task or algorithm on P processors. An ideal situation is that where the value of S_p is directly proportional to P and $E_p = 1$. In practice, however, we often have to content ourselves with efficiency factors of only several percent.

Let us note that if the portion of consecutive operations is equal to θ, then the maximum speedup is defined by Amdahl's law [16]:

$$S_p = T_1/(\theta T_1 + (1 - \theta T_1))/P = P/[1 + \theta(P - 1)].$$

The development of supercomputer technologies occurs in two main directions: high-performance computing (HPC) and operations with Big Data. Note that the convergence of these two trends (intensive data computing) has recently been observed. On the whole, the evolution of MPS generations and extremum modeling problems is accompanied by a similar growth of RAM speed and capacity (the number of teraflops or petaflops is quite similar to the number of terabytes or petabytes of the computer).

The main objective of programming parallel algorithms is to minimize the information exchange, since the total problem time T equals the sum of two terms:

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

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

It is evident from all the above that the notion of the quality of algorithms changes for large tasks: from any two methods being compared, the best is not the one that requires fewer computations but the one that is executed faster on MPSs of the type under consideration. In other words, there appears a new concept of computation optimization based on the search for approaches that would significantly reduce the volume of data transferred between processors, even if they increase the number of arithmetic operations.

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

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 macro-objects, which should not be too many (tens, hundreds or, at worst, thousands). Therefore, it seems desirable to manage without exchanges, copying the computations in all MPI processes and storing in them the geometric and functional data structures obtained.

The mesh generation may formally be represented as a data transformation: GDS + FDS $\rightarrow$ MDS. Note that the mesh data structure for the entire computational space may have a large volume. For this reason, it is natural to create an MDS by each MPI process for "its" mesh subdomain (with a certain overlapping). The formation of distributed data at the initial stage is reasonable,

so much 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, namely locally by a subdomain and globally. Decomposition problems can employ two tactics: subdomain construction, which precedes mesh generation (for example, it is natural to separate media with contrasting material properties), or direct formation of mesh subdomains. We should also bear in mind that many efficient algorithms are based on special re-arrangements of components (one may speak of such tasks in terms of graphs as well), and all the respective procedures should be accessible to all MPI processes or subdomains, which will, on the whole, substantially reduce data exchange. The popular software packages METIS, parMETIS and other tools for graph partition are effective re-arrangement instruments (see, for example, the review in Algowiki [9]).

Moving boundary problems are the most computationally complex, since they imply that adaptive grids are dynamically reconfigurable as well. Many efficient methods are based on a local refinement and multigrid approaches, whose instrumental support should also be distributed.

Upon obtaining the distributed data arrays, mapping onto MDS, GDS and FDS, one can approximate the original problem in parallel. For this purpose, FEM and FVM have a unique technology for computing local matrices and assembling a 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. Therefore, this stage can be perfectly implemented without exchanges. The principal operations performed by the mesh cells independently of each other can be effectively parallelized using multi-thread computing. In non-stationary problems and also in nonlinear or optimization computations, approximations are repeated. However, from the point of view of adaptation to computing devices, this usually changes nothing.

Linear systems are the most important intermediate elements when solving algebraic problems. Owing to this, we will focus on them. Special attention should be given to very large SLAEs with sparse matrices, which emerge after the FEM- or FVM-assisted approximation of differential or respective variational multi-dimensional problems on unstructured grids. From the point of view of the classification of algorithms, the SLAEs to be solved can be divided into two major classes: special and general ones. For the former, which comprise systems occurring in boundary value problems with separable variables, there are superfast direct and/or iterative problem solvers, as the fast Fourier transform or alternating direction implicit (ADI) methods with optimal sets of iterative parameters. These approaches have been in demand over the last decades, because of practical requirements to solve the actual Lyapunov and Sylvester matrix equations.

Direct methods for large sparse SLAEs of the general type are actively improving; however, in the most advanced versions of the popular PARDISO [14] and MUMPS programs, their applicability is limited, mainly because of their

requirements on RAM volume. Iterative additive domain decomposition methods (DDMs) constitute the main tool for a parallel highly productive solution of SLAEs of this type. DDMs are covered in a considerable body of specialized literature (see, for example, the review in [17]), and have been discussed at 23 major international conferences devoted to this topic. The essence of these decomposition methods consists in dividing a computational mesh domain into subdomains with parametrized overlapping (in a 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 block Jacobi method, which leads to 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-grid correction, low-rank matrix approximations, which are implemented in the library KRYLOV [15]. A systematic analysis of modern approaches in algebraic DDMs is presented in [18]. As examples of well-known libraries for parallel solving SLAEs, we can mention PETSc [4] and pARMs [19].

Parallel time integration methods for solving evolution problems are a special topic of interest. An overview of the exciting and rapidly developing area of parallel time algorithms is given in [20].

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 multi-threaded computations are performed using OpenMP in common memory. Note that a substantial acceleration is achievable if inter-processor exchanges are matched with synchronous performance of arithmetic operations in subdomains. A separate problem is how to effectively use universal graphics accelerator cards with a great number of computer cores but relatively slow communications (General Purpose Graphic Processor Units, GPGPU), as well as Intel Xeon Phi units and advanced Field Programmable Gate Arrays (FPGAs).

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

A special topic of parallelization analysis is that of optimization approaches to solving inverse problems. Some computational issues of this important area are described in [21]. Usually, the solution of an inverse problem requires solving successively a set of direct tasks. In this case, the speedup of parallel computing does not change. Exceptions must be made for the search for several minima of the goal functional and the solution of a global minimization problem. In these cases, we can decompose the domain in the spaces of parameters that should be determined in the original problem, and find an auxiliary inverse constrained subproblem independently in each subdomain.

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

From the point of view of large-scale parallelization, optimization methods and computational experimentation control are a superstructure over data-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.

4 Conclusions

From the previous analysis of computational models, algorithms and technologies, we can conclude that the infrastructure of large-scale mathematical modeling constitutes a sufficiently large and complicated system. Also, the optimal control of parallel computing requires a careful analysis of the peculiarities of every technological stage. Tactics and strategies of scalable parallelism can be different in terms of both the algorithm and the total task to be solved. We want to make two more comments. First, creating a high-performance integrated numerical environment for solving a wide class of applications on heterogeneous supercomputers with distributed and hierarchical shared memory is a big management problem, which can be solved on the base of common component architecture (CCA) principles (see a discussion in [22]). And second, the implementation of large mathematical experiments should be actually done in a cloud computing framework, with the task-flow technologies at Data Center, and here we have another view to parallelism problems. It is possible to analyze the optimization statement for the runtime, the performance or the speedup in terms of one algorithm, a particular technological stage or a concrete applied mathematical problem. In a sense, we obtain at different levels various local constrained minimization or global multi-variable minimization problems, and the final strategy solutions will be different. However, the main objective of this paper has just been to outline some issues, while the real solutions of these subjects are topics for a special additional research.

References

1. ANSYS - Simulation Driven Product Development. http://www.ansys.com
2. Logg, F., Mardal, K.-A., Wells, G.N. (eds.): Automated Solution of Partial Differential Equations by Finite Element Method: The FEniCS Book. Springer, Heidelberg (2011)
3. Schoberl, J.: Netgen - an advancing front 2D/3D mesh generator based on abstract rules. Comput. Vis. Sci. **1**, 41–52 (1997). doi:10.1007/s007910050004
4. PETSc. http://www.mcs.anl.gov/petsc
5. Open FOAM - The Open Source Computational Fluid Dynamics (CFD) Toolbox. http://www.open-foam.com
6. DUNE Numerics. Distributed Unified Numerical Environment. http://www.dune-project.org
7. Il'in, V.P., Skopin, I.N.: Computational programming technologies. Programm. Comput. Softw. **37**(4), 210–222 (2011). doi:10.1134/s0361768811040037
8. Ilin, V.P.: Fundamental Issues of mathematical modeling. Herald Rus. Acad. Sci. **86**, 118–126 (2016). doi:10.1134/s101933161602009x
9. AlgoWiki: Open encyclopedia of algorithm properties. http://algowiki-project.org
10. Il'in, V.P., Skopin, I.N.: About performance and intellectuality of supercomputer modeling. Program. Comput. Softw. **42**, 5–16 (2016). doi:10.1134/s0361768816010047
11. Golubeva, L.A., Il'in, V.P., Kozyrev, A.N.: Programming technologies in geometric aspects of mathematical modeling. Vestnik NGU. Seria: Inf. Tekhnol. **10**, 25–33 (2012). (in Russian)
12. Il'in, V.P.: DELAUNAY: A technological environment for mesh generation. Siberian Zhurnal Indus. Math. **16**, 83–97 (2013). (in Russian)
13. Butygin, D.S., Il'in, V.P.: Chebyshev: Principles of automation of algorithm construction in an integrated environment for mesh approximations of initial boundary value problems. In: Proceedings of International Conference on Parallel Computational Technologies 2014, Izd. YuUrGU, Chelyabinsk, pp. 42–50 (2014). (in Russian)
14. Intel Math. Kernel Library. http://software.intel.com/en-us/intel-mkl
15. Butyugin, D.S., Gurieva, Y.L., Il'in, V.P., et al.: Functionality and technologies of algebraic solvers in Krylov's library. Mathematical Modeling, Programming & Computer software, pp. 76–86. Bulletin of south Ural state university, Russain Federation (2013). (in Russian)
16. Konshin, I.N.: Parallel Computational Models to Estimate an Actual Speedup of Analyzed Algorithm. Russian Supercomputing Days: Proceedings of the International Conference, pp. 269–280. MSU Publ. (2016) (in Russian). doi:10.1007/978-3-319-55669-7_24
17. Dolean, V., Jolivet, P., Nataf, F.: An Introduction to Domain Decomposition Methods: algorithms, theory and parallel implementation. doi:10.1137/1.9781611974065.. https://archives-ouvertes.fr/cel-01100932
18. Il'in, V.P.: Problems of parallel solution of large systems of linear algebraic equations. J. Math. Sci. **216**, 795–804 (2016). doi:10.1007/s10958-016-2945-4
19. Saad, Y., Sosonkina, M.: pARMS: A package for the parallel iterative solution of general large sparse linear systems user's guide. Report UMSI 2004–8, Minnesota Supercomp. Inst., Univer. of Minnesota, MN (2004)
20. Gander, M.J.: 50 Years of Time Parallel Time Integration. doi:10.1007/978-3-319-23321-5_3. http://www.unige.ch/∼gander/Preprints/50YearsTimeParallel.pdf

21. Il'in, V.P.: Numeric solving of direct and inverse problems of electromagnetic prospecting. Sibirian Zhurnal of Vychislitelnoi Mathematiki **6**, 381–394 (2003). (in Russian)
22. Il'in, V.P.: Component technologies of high-performance mathematical modeling. In: Proceedings of the International Conference on Parallel Computational Technologies - 2015 (UFU, IMM UrO RAN, Yekaterinburg), pp. 166–171 (2015). (in Russian)