Parallel Numerical Methods Course for Future Scientists and Engineers

Iosif Meyerov, Sergey Bastrakov, Konstantin Barkalov, Alexander Sysoyev, and Victor Gergel(✉)

Lobachevsky State University of Nizhni Novgorod, Nizhni Novgorod, Russia
{iosif.meyerov,sergey.bastrakov,konstantin.barkalov,
alexander.sysoyev,victor.gergel}@itmm.unn.ru

Abstract. The rise of computational science has facilitated rapid progress in many areas of science and technology over the last decade. There is a growing demand in computational scientists and engineers capable of efficient collaboration in interdisciplinary groups. Training such specialists includes courses on numerical analysis and parallel computing. In this paper we present a new Master's course Parallel Numerical Methods which bridges the gap between theoretical aspects of numerical methods and issues of implementation for modern multicore and manycore systems. The course aims to guide students through the complete process of solving computational problems, from a problem statement to developing parallel software and analyzing results of computational experiments. An important feature is that many of practical classes are based on research done at the HPC Center of the University of Nizhni Novgorod and therefore illustrate issues, which students may encounter in their research and future career.

Keywords: Education in computational science · Numerical analysis · Parallel computing · Master's program

1 Introduction

The importance and relevance of modern methods of computational science can hardly be overestimated. The progress in development of computer systems and applications for solving scientific and technical problems confronts more and more new ambitious challenges to scientists and engineers. In many fields there is a demand for non-ordinary solutions which allow replacing natural experiments with computational ones, therefore essentially shortening the way from an innovative idea to its technological implementation. Among such fields are computer-aided design, computational physics, computational biomedicine and others. These areas can greatly benefit from collaboration of experts in different areas: researchers in natural and social sciences, theoretical and applied, mathematicians, and software engineers. However, efficient collaboration in such multidisciplinary groups is not always easy, as different professional communities tend to have specific traditions, methods and terminology.

V. Voevodin and S. Sobolev (Eds.): RuSCDays 2017, CCIS 793, pp. 3–13, 2017.
https://doi.org/10.1007/978-3-319-71255-0_1

A way to approach this issue is to train specialists oriented towards multidisciplinary collaboration as a part of Master's programs. The institute of IT, mathematics and mechanics at the Lobachevsky State University of Nizhni Novgorod (UNN) has created a Master's program in computational science, which includes a wide range of topics concerning numerical simulation, applied mathematics, computational mathematics, and computer science. Many students on this program are members of multidisciplinary groups carrying out research projects at the UNN HPC center [1]. By the time of graduation these students have some real-world experience in computational science, which can be valuable for their career.

This paper describes a core course in our Master's program in computational science, Parallel Numerical Methods. To date, a considerable amount of educational and methodical literature on numerical methods is available, for example, the latest editions of the classical textbooks [2–4]. In the literature on numerical methods, the issues of development, application and theoretical substantiation of algorithms for numerical solution of various classes of mathematical problems are considered. Courses on theoretical aspects of numerical methods have been developed for decades with lots of excellent courses and materials available. Parallel programming, performance analysis and optimization are much more rapidly developing areas. Evolution of hardware, tools and technologies constantly creates new challenges and requires development and modernization of course materials. There are respectable textbooks on key technologies for parallel programming, for example, [5, 6]. Some books consider optimization of applications from various areas for modern architectures [7–9]. Our Parallel Numerical Methods course aims to guide students through the complete process of solving computational problems, from a problem statement to developing parallel software and analyzing results of computational experiments. The course forms skills in studying a problem at hand and its mathematical model, choosing appropriate numerical methods, developing a parallel algorithm and its implementation for multicore and manycore systems, performing computational experiments and analyzing results in terms of accuracy and performance. An important feature of the course is that most problems considered are based on the experience from research projects done at the UNN HPC Center. These examples illustrate the common issues, which students are likely to encounter in their future career.

This paper is organized as follows. Section 2 contains a short overview of courses on numerical analysis and numerical methods. Section 3 presents the main ideas and principles of our Parallel Numerical Methods course. Course structure is described in Sect. 4 with examples of lectures and practical classes given in Sect. 5. Section 6 is devoted to assessment of student performance. Section 7 concludes the paper.

2 Related Work

Courses on numerical analysis and numerical methods, for example [10–14] are delivered in many universities worldwide. The textbooks with several editions released, including [15, 16], form a methodical basis for such courses. In general, these are mostly

classical courses on numerical methods with the main focus on theoretical material: theorems on approximation, stability and convergence.

There are also courses which cover the classical topics of numerical methods and are directed particularly towards the issues of implementation for modern computational systems. A notable example is the Introduction to Numerical Methods course at MIT [10]. The course begins with considering the issues of performance, software optimization, and floating-point arithmetic. Then, the basic numerical algorithms of linear algebra (solving the eigenvalue problem, direct and iterative methods for solving systems of linear equations) are considered. There are several advanced courses concerning parallel aspects of numerical algorithms, most notably in linear algebra [17, 18]. Linear algebra problems are rather intuitive, and, therefore, very suitable to demonstrate the basics of parallel computing. Other numerical methods are typically part of courses on scientific computing, for example [19–21].

This paper presents the Parallel Numerical Methods course developed at the UNN HPC Center based on 15 years' experience of research in computational science. The course covers numerical methods and issues of parallel implementation for a wide range of problems: dense and sparse linear algebra, direct and iterative solvers, finite-difference schemes for ordinary and partial differential equations, Monte Carlo methods.

3 Course Description

The Parallel Numerical Methods course described in this paper is a core course of the Master's program in computational science at the Lobachevsky State University of Nizhni Novgorod. The goals of the course are mastery of numerical algorithms and considering the issues of implementation, performance and scalability on modern hardware. The course covers parallel aspects of the classical topics of numerical methods, including dense and sparse linear algebra, ordinary and partial differential equations, Monte Carlo methods.

Course prerequisites include fundamentals of linear algebra, mathematical analysis, numerical methods, and parallel programming. This set of skills is rather typical for graduates of Bachelor's programs in applied mathematics and computer science, such as [22]. Since some students with a solid mathematical background may not be familiar with parallel programming, for example, Bachelor's in mathematics, our curriculum offers an optional parallel programming course in the same semester, which completely covers demands of the Parallel Numerical Methods course.

The course is based on the following main principles:

1. *A wide range of topics*: the course covers basic topics of numerical methods, widely used for scientific and engineering computing in various areas.
2. *Balance between numerical analysis and computer science*: the course combines mathematically strict presentation of material with proper attention to efficient implementation for parallel hardware.
3. *Integrity*: the course demonstrates the whole chain of stages required to solve a computational problem (problem statement, mathematical model, serial algorithm,

parallel algorithms, implementation and parallelization, computational experiment and analysis).
4. *Real-world experience*: demonstrating approaches used by research groups to solve state-of-the-art problems of computational science.
5. *Assessment based on applications*: assessment of student performance is done based mostly on ability to solve a problem going through all stages, from problem statement to computational experiment and analysis.
6. *Flexibility*: the course is designed in such a way that modules/practical classes are to a large degree independent.

Based on the above mentioned principles and course prerequisite we have decided to give basic mathematical statements and theorems in the lectures without proofs, making references to textbooks on numerical methods. Most lectures combine theoretical descriptions of methods with approaches to parallel implementation and demonstrations of performance results. Each practical class is a detailed study and development of a parallel implementation for a computational problem, using tools part of Intel Parallel Studio (C++ Compiler, Cilk Plus, TBB, MKL, Amplifier). The course contains a large number of case studies demonstrating applications from computational physics, computational finance, computational biology, and other areas.

4 Course Outline

Below we give a list of basic modules of the course with brief descriptions.

1. *Elements of computer arithmetic*. The topic of this module is representation of floating point numbers in computer memory [23]. The problems of computational error accumulation and methods for its reduction and control are discussed. Typical examples, where error accumulation may result in incorrect computation results, are presented.
2. *Direct methods for solving systems of linear equations*. This module is devoted to direct methods of solving systems of linear algebraic equations: Gaussian elimination, Cholesky decomposition, Thomas and reduction methods. The classical methods are presented and estimates of complexity given. We demonstrate insufficient efficiency of naïve implementations of these methods on modern computational architectures. The idea of block data processing is highlighted consistently. The problems of sparse algebra are considered here as well. A brief review of the data structures for storing sparse matrices is given, typical problems arising when performing the basic operations with sparse matrices are considered. Comparison of the matrix-vector and matrix-matrix multiplication algorithms for the cases of dense and sparse matrices is given. Cholesky decomposition is considered as an example of a more complex computational algorithm for sparse matrices. The issue of increasing amount of nonzero elements after factorization is demonstrated, several algorithms of matrix reordering to reduce the fill-in of the resulting matrix (minimum degree and nested dissection methods) are presented.

3. *Iterative methods for solving systems of linear equations*. This module considers iterative methods for solving the systems of linear equations, from the basic methods (simple iteration, Jacobi, Seidel, upper relaxation methods) to Krylov-type methods (generalized minimal residual, conjugated and biconjugated gradient methods). We discuss approaches to parallelization, give theoretical and experimental estimates of speed-up. The module also covers some methods of preconditioning: the basic methods (Jacobi method, Gauss-Seidel method) and the methods based on the incomplete LU-decomposition (ILU(0) and ILU(p) factorization).
4. *Methods for solving ordinary differential equations*. This module concerns the basic methods for solving ODEs: Runge-Kutta methods and Adams methods. The parallel variants of the methods for solving systems of ODEs are considered. For Runge-Kutta methods, the pipelining scheme of solving systems of ODEs with a sparse right-hand part is given. Solving a system of ODEs arising from simulating a neural system is considered as an illustrative example.
5. *Methods for solving differential equations in partial derivatives*. The module encompasses the issues of parallel solving differential equations in partial derivatives. Typical equations in partial derivatives (of hyperbolic, parabolic, and elliptical types) are considered. The finite differences method is delivered to the students as a method of reduction of differential equations to algebraic ones, leading to solving the difference equations. The explicit and implicit schemes of solving parabolic and hyperbolic equations and issues of parallel implementation are considered. The advantages and drawbacks of each approach are discussed. The pentadiagonal system of linear equations arising while solving 2D Poisson equation is discussed separately. The wave scheme of data processing in parallel solving of this system by iterative methods is presented.
6. *Monte Carlo methods*. This module introduces general concepts of the Monte Carlo methods. It describes issues of utilizing pseudo-random number generators in parallel programs, ways of reducing variance and presents applications for multidimensional integration, computational physics, and computational finance.

5 Conducting the Classes

The lecture part of the course concerns construction and analysis of efficient parallel algorithms from various topics of numerical methods. The presentation is accompanied by the results of the computational experiments and analysis. For example, a lecture on Cholesky factorization of a dense matrix is organized as follows. We start with the definition of Cholesky factorization and describe applications for solving systems of linear equations with a symmetric positive-definite matrix. Then we show how a naive algorithm can be constructed based on the definition and estimate its complexity. Approaches to parallelization are considered and scaling efficiency obtained for our implementation is demonstrated. We proceed to estimating cache efficiency of the naive algorithm and introducing the idea of blocking to increase cache reuse. Serial and parallel block Cholesky factorization algorithms are presented along with performance and

scaling efficiency of our implementation compared to the naive algorithm. Analysis of the results concludes the lecture.

Another section of the course covers sparse linear algebra algorithms. One of the lectures considers sparse direct solvers. We discuss advantages and disadvantages of direct methods, give a general computational scheme, review main approaches to parallelization of sparse matrix factorization, and introduce widely used software. The demonstration is done using the open source sparse matrix reordering library PMORSy [24] developed at the UNN HPC Center. The example of workload distribution during a sparse matrix reordering is shown below (Fig. 1).

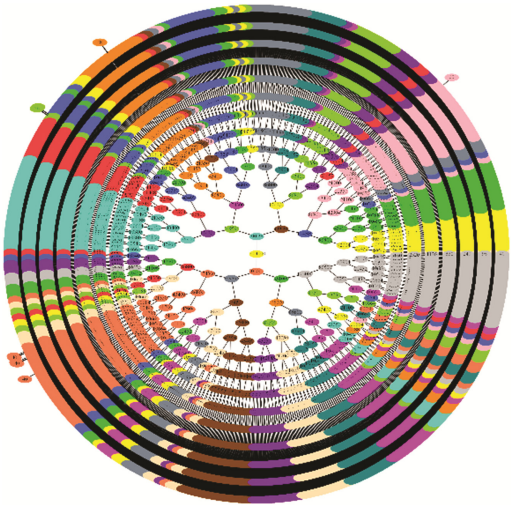

Fig. 1. Task mapping for a test matrix on 16 threads. Logical tasks are nodes of the graph, dependencies between them are edges. Descendant nodes correspond to the tasks generated after the parent task is completed. Same colored nodes are processed by the same thread [24].

Each practical class is a study of a selected computational problem. A problem description includes a problem statement, brief information on the research area, numerical method, possible approaches of parallelization, analysis of correctness, performance and scaling efficiency, and possible ways to improve it. The class is conducted either in form of a demonstration and analysis done by a teacher or in form of students gradually developing and analyzing their implementation following the description.

Let us describe several practical classes, which are part of the course. One group of classes is based on research done by a group of mathematicians and computer scientists on computational finance. A feature of this area is that problems are often seemingly simple; however the models and methods used are rather complicated and rely on statistics, differential equations and mathematical optimization. Nevertheless, all formalisms used have a clear financial interpretation, which makes it easier to introduce financial terms while keeping the material mathematically strict. Some of the methods used in computational finance can also be applied for other areas. Below we describe two concrete examples of this group.

The first example is performance optimization of Black-Scholes pricing. The problem is to calculate the fair prices for a set of European options [25], which is a fairly simple problem of financial mathematics. In this case, the result can be calculated analytically. From the programming point of view, this is a trivial problem (just to apply a formula for input data); however, it demonstrates that the computational time can vary by an order of magnitude even in such a simple program depending on programming and optimization skills and techniques. First, we introduce a model and basic concepts of a financial market and some intuitive descriptions of the option pricing problem briefly. We create a basic implementation, analyze its performance and improve it in a step-by-step fashion: eliminate unnecessary type casts, carry out invariants, perform mathematical transforms that replace heavy math routines with the lighter ones, vectorize and parallelize, perform warm-up to reduce overhead on thread creation, try reducing precision of floating-point operations, utilize streaming stores. The effects of these optimization techniques are demonstrated on both CPU and Intel Xeon Phi. The main methodological direction of this work is to teach pragmatics of using mathematical routines (choosing efficient mathematical library, controlled reduction of precision if justified), vectorization by compiler directives and optimization for manycore architectures. The detailed description of this work is published in [25].

The second example on computational finance is performance optimization of Monte Carlo option pricing (Fig. 2). We consider the case where the fair prices cannot be computed analytically. A widely used method is Monte Carlo simulation, which is relatively easy to implement and has a huge degree of parallelism. We cover topics of correct pseudo-random number generation in parallel applications and demonstrate typical errors in this area. Efficiency of low-discrepancy sequences and approaches to parallel implementation are shown. We demonstrate methods to check accuracy of a Monte Carlo simulation. The main value for students is to learn how to correctly use pseudo-random number generators in parallel programs.

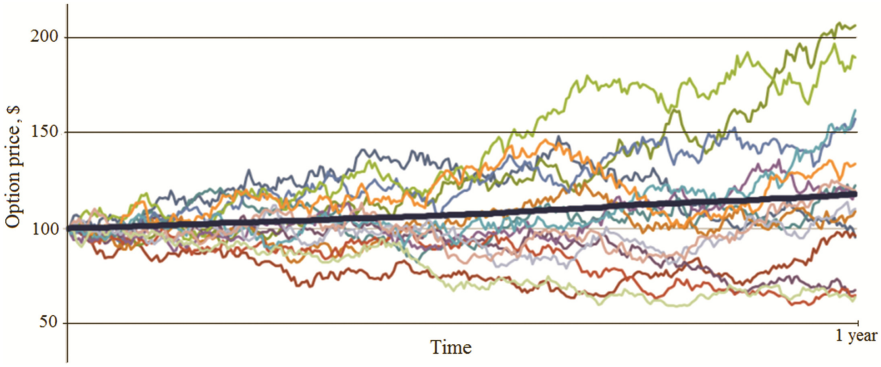

Fig. 2. Evolution of option price in time. Several Monte Carlo trajectories and the average are shown.

Another group of practical classes is devoted to computational physics. One example is based on a research project in plasma physics done by a large group of theoretical and

experimental physicists, mathematicians and software developers from the UNN HPC Center, Institute of Applied Physics of Russian Academy of Sciences and Chalmers University of Technology. The example concerns solving Maxwell's equations in 3D space using the Finite Difference Time Domain method, a cell of the grid used is given at Fig. 3. We discuss choosing data layout, vectorization, scaling efficiency on Intel Xeon Phi. Another example is Monte Carlo simulation of brain sensing by optical diffuse spectroscopy based on a joint research by the UNN HPC Center and Institute of Applied Physics. We show problem statement and demonstrate results of a straightforward implementation of Monte Carlo simulation. Using a profiler, we show an approach to change data structures in order to improve memory efficiency and load balancing on Xeon Phi. The methodical value of these two examples is to demonstrate a pragmatic choice of data structures and approaches to load balancing on many-core architectures.

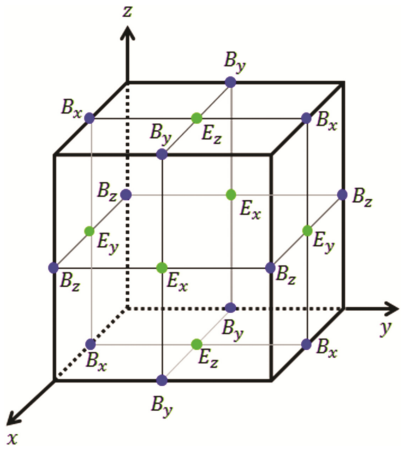

Fig. 3. A cell of the spatial grid used in the Finite Difference Time Domain method.

6 Assessment of Student Performance

As a very basic form of assessment, all students pass online testing on every module of the course. While useful for monitoring the current progress, it only focuses on theoretical knowledge, not practical skills. Thus, the main form of assessment is solving one or several computational problems. We believe it is a better form of assessment since it covers the whole cycle of computational scientist work: studying methods, creating a serial implementation, verifying its correctness, creating a parallel implementation, optimizing its performance and scaling efficiency.

There are currently three groups of test problems:

1. Block algorithms for dense linear algebra problems (e.g. block LU and Cholesky factorization).
2. Iterative solvers for sparse systems of linear equations (e.g. conjugate gradient method).

3. Solvers for ODEs and PDEs (e.g. finite-difference schemes).

Each student is randomly assigned one problem from each group to provide a good coverage of the course material.

For each problem we provide sets of parameters to be used for testing and requirements for performance and scaling efficiency. An implementation is accepted once it passes all tests in terms of correctness, performance and scaling efficiency.

On a technical side, we use an automated checking system based on the open source edge software (https://ejudge.ru/). Students upload source code files and a make file via a web interface. The system builds the submitted code, runs it on all test items, and checks correctness and performance. The specific way of checking the correctness depends on the type of a problem. For example, for the direct methods for solving systems of linear equations, the checking is performed by substitution (with a tolerance depending on a norm of the matrix); for the methods for solving differential equations, the accordance of behavior of the error when increasing the grid dimensionality to the theoretical properties of the methods is checked. The efficiency of parallel implementations is checked by means of comparing the speed-up relative to the sequential version with the threshold value dependent on the problem.

Let us present two examples of problem statements.

Example 1. *The conjugate gradient method for sparse systems of linear equations.*

Problem statement: Implement the conjugate gradient method for solving a sparse symmetric system of linear equations in form $Ax = b$. Choose an exact solution x^* and use Ax^* as the right-hand side. Use the norm of the difference between the consecutive approximations as the stop condition. Output the norm of the residual on the last step of the method.

Input format: a sparse symmetric matrix in .mtx format (from the University of Florida Sparse Matrix Collection).

Output format: a single number that is the norm of the residual on the last step of the method.

Verification: checking that the norm of the residual does not exceed 1% of the norm of the input matrix.

Limitations of the problem size: no more than 100 000 000 non-zero elements in the input matrix.

Requirements on scalability: the scaling efficiency is not less than 50%.

Example 2. *The Crank-Nicolson method for solving the 1D heat equation.*

Problem statement: Implement the Crank-Nicolson method for solving the 1D dynamic heat equation with the Dirichlet boundary conditions. Choose a non-trivial function as the exact solution and construct the right-hand side, initial and boundary conditions accordingly. Use the cyclic reduction method for solving the resulting system of linear equations with a tridiagonal matrix.

Input format: the number of grid nodes on space and time axes.

Output format: the maximum difference between the exact and numerical solutions on the grid.

Verification: steadily increasing the number of grid nodes checking that the error is proportional so the sum of squares of space and time steps.

Limitations of the problem size: the total number of grid nodes is no more than 1 000 000.

Requirements to scalability: the scaling efficiency is not less than 40%.

7 Conclusion

This paper describes the Parallel Numerical Methods course for Master's program in computational science at the Lobachevsky State University of Nizhni Novgorod. The main goal of the course is to bridge the gap between theoretical aspects of numerical methods and issues of implementation for modern multicore and manycore systems. This is an important chain in training specialists capable of working in multidisciplinary scientific and engineering groups. The course relies on basic knowledge of numerical methods and parallel programming obtained during Bachelor's programs and concentrates of parallelization and efficiency.

The course has a flexible modular structure. Each module is devoted to a key area of numerical methods. Most lectures demonstrate a whole cycle from a mathematical model to results of computational experiments in terms of accuracy and efficiency. Most practical classes are devoted to solving computational problems in different areas. An important feature is that many of practical classes are based on research done at the UNN HPC Center and therefore illustrate issues, which students may encounter in their research and future career. Assessment of student performance is mostly done based on solving test computational problems. By means of an automated system, we control accuracy of submitted solutions as well as performance and scaling efficiency.

Course materials are currently available in Russian on the website http://www.hpcc.unn.ru/?doc=491. These materials have been used for several training programs for teachers and researchers. Over 500 students have been trained since 2012. There is an ongoing process of extending the materials and translating them to English; a preliminary English version of materials for some modules is available at http://hpc-education.unn.ru/en/trainings/collection-of-courses. Another direction of future work is creating a course for one of the widely used e-learning systems.

References

1. Bastrakov, S., Meyerov, I., Gergel, V., Gonoskov, A., Gorshkov, A., Efimenko, E., et al.: High performance computing in biomedical applications. Procedia Comput. Sci. **18**, 10–19 (2013)
2. Stoer, J., Bulirsch, R.: Introduction to Numerical Analysis, vol. 12. Springer, Heidelberg (2013)
3. Mathews, J.H., Fink, K.D.: Numerical Methods Using MATLAB, vol. 31. Prentice hall, Upper Saddle River (1999)
4. Hamming, R.: Numerical Methods for Scientists and Engineers. Courier Corporation (2012)
5. Andrews, G.R.: Foundations of Parallel and Distributed Programming. Addison-Wesley Longman Publishing Co. Inc., Boston (1999)

6. Prasad, S.K., Gupta, A., Rosenberg, A.L., Sussman, A., Weems, C.C.: Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses, 1st edn. Morgan Kaufmann, San Francisco (2015)
7. Jeffers, J., Reinders, J. (Eds.): High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, 1st edn. (2014)
8. Jeffers, J., Reinders, J., Sodani, A. (Eds.): Intel® Xeon Phi™ Processor High Performance Programming, Knights Landing Edition (2016)
9. Hwu, W.-M.W. (Ed.): GPU Computing Gems Jade Edition. Morgan Kaufmann (2011)
10. Introduction to numerical methods (2010). Accessed Jan 2017. MIT Open Courseware: http://ocw.mit.edu/courses/mathematics/18-335j-introduction-to-numerical-methods-fall-2010
11. Introduction to numerical analysis (2004). http://ocw.mit.edu/courses/mathematics/18-330-introduction-to-numerical-analysis-spring-2004
12. CME206 – Introduction to Numerical Methods for Engineering (2016). http://scpd.stanford.edu/search/publicCourseSearchDetails.do?method=load&courseId=11683
13. Math 128A: Numerical Analysis (2014). Accessed Jan 2017. http://persson.berkeley.edu/128A
14. Course MAT321 Numerical Methods (2014). Accessed Jan 2017. https://www.math.princeton.edu/undergraduate/course/MAT321
15. Burden, R., Faires, J.: Numerical Analysis, 9th edn. Brooks-Cole, Boston (2010)
16. Kincaid, D., Cheney, E.: Numerical Mathematics and Computing, 7th edn. Brooks-Cole, Boston (2012)
17. Demmel, J.: Matrix Computations/ Numerical Linear Algebra (2016). https://people.eecs.berkeley.edu/~demmel/ma221_Spr16
18. Saad, Y.: Computational Aspects of Matrix Theory, Sparse Matrix Computations (2015). http://www-users.cs.umn.edu/~saad/teaching.html
19. Dongarra, J.: Scientific Computing for Engineers: Spring 2012 (2012). http://www.netlib.org/utk/people/JackDongarra/WEB-PAGES/SPRING-2012/cs594-2012.htm
20. Heath, M.: Parallel numerical algorithms (2013). http://www.mat.unimi.it/users/pavarino/heath_2013
21. Edelman, A.: Numerical Computing with julia (2016). http://courses.csail.mit.edu/18.337/2016/calendar.html
22. Gergel, V., Liniov, A., Meyerov, I., Sysoyev, A.: NSF/IEEE-TCPP curriculum implementation at University of Nizhni Novgorod. In: Proceedings of Fourth NSF/TCPP Workshop on Parallel and Distributed Computing Education, pp. 1079–1084 (2014)
23. Muller, J.M., Brisebarre, N., De Dinechin, F.: Handbook of Floating-Point Arithmetic. Springer (2009)
24. Pirova, A., Meyerov, I., Kozinov, E., Lebedev, S.: PMORSy: parallel sparse matrix ordering software for fill-in minimization. Optim. Method. Softw. **32**, 274–289 (2016)
25. Meyerov, I., Sysoyev, A., Astafiev, N., Burylov, I.: Performance optimization of Black-Scholes pricing. In: Jeffers, J., Reinders, J. (Eds.) High Performance Parallelism Pearls: Multicore and Many-core Programming Approaches, pp. 319–340 (2014)