International Journal of Engineering Education Vol. 34, No. 4, pp. 1250–1260, 2018
Printed in Great Britain
0949-149X/91 $3.00+0.00
© 2018 TEMPUS Publications.

Key Issues of Low-Level Parallel Programming in the Individual Projects for Graduate Students*

IRINA ZAKHAROVA
Software Department, Institute of Mathematics and Computer Science, University of Tyumen, Volodarskogo 6, 625003 Tyumen, Russian Federation. E-mail: i.g.zakharova@utmn.ru

ALEXANDER ZAKHAROV
Information Security Department, Institute of Mathematics and Computer Science, University of Tyumen, Volodarskogo 6, 625003 Tyumen, Russian Federation. E-mail: a.a.zakharov@utmn.ru

This paper presents the research project based methodology of teaching parallel programming to master's students in a High Performance Computing program. The requirements for completing a master's degree state that all students should be able to develop computer simulation programs using parallel and distributed computing technologies, regardless of students' background and their preferences for in-depth study of high or low-level programming, administration, and information security. Creating computer simulations based on high-performance computing is impossible without the experience of solving such key issues of low-level parallel programming as the data flow management, synchronization, load balancing and fault tolerance. We believe that the best way to explore these issues is phased implementation of appropriate algorithms in the application, and then carrying out computational experiments. Therefore, as a main tool for the practical study, we offer the implementation of special project tasks. While developing the course tasks, we have used not only our teaching experience of parallel programming for undergraduate and graduate students, but we also relied on the existing practice of the development of distributed computing systems. In addition to the classic tasks, students explored pairing algorithms, load balancing and fault tolerance through implementation in distributed applications and testing in computational experiments. Our experience has shown that this approach to teaching parallel programming, which includes modeling and simulations, enabled students to proceed gradually from classic tasks to the implementation of full-scale research projects.

Keywords: parallel programming; graduate course; load balancing; fault tolerance; computational experiments

1. Introduction

Undergraduate courses in High Performance Computing (HPC) and Parallel and Distributed Computing (PDC), and in particular, parallel programming, are no longer something exotic in universities that respond to rapid changes in the IT field. What was used 20 years ago only for solving tasks on supercomputers, now works in applications for smartphones. That is why it is a basic expectation within the IT community that any professional software developer should be able to use the technology of parallel programming. The current requirements for training programmers, formulated in the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing [1, 2], suggest that undergraduate IT programs, specifically Computer Science (CS) and Software Engineering (SE) must include Courses in PDC. In Russia, the Code of Knowledge and Skills for supercomputer education has been developed based on the Curriculum Initiative on Parallel and Distributed Computing [3, 4]. This document contains recommendations for universities, which, first of all, must follow the "State Educational Standards" and professional standards. However, of all existing professional IT standards [5], only the professional standard "System programmer" [6, p. 5] reflects in its most general form the relevant competences (PDC basics knowledge).

Therefore, in the context of studying parallel programming for Russian educational programs, many questions remain open. Unfortunately, even graduates of IT fields of the bachelor's program of Russian universities sometimes have only superficial ideas about PDC concepts.

At the same time, the master's programs in the HPC area tend to be oriented toward students with sufficient knowledge of algorithms and technologies for parallel programming. However, the mobility of students, combined with the rights of universities themselves to determine the content of curricula in the absence of training standards create certain teaching difficulties.

In this paper, we propose a method of teaching parallel programming based on students' individual projects. We show that this methodology helps students not only to learn the basics of parallel programming, but also prompts them to investigate such complex problems as load balancing and fault tolerance. Based on our experience in using this method in teaching, we suggest that the following

* Accepted 1 February 2018.

conditions are necessary for obtaining successful results: (1) a unified structure of tasks; (2) various levels of complexity for tasks; (3) breakdown of the project into separate small steps; (4) specific deadlines and requirements for interim results; (5) the opportunity for students to choose the topic and the level of complexity of the task for the project. Students choose the level of complexity along with the conditions that the solution of the original problem should satisfy (however, the level of complexity and completeness of conditions cannot be lower than the minimum requirements for the project).

1.1 Background

Institute of Mathematics and Computer Science at Tyumen State University offers a two-year master's program in High Performance Computing. It focuses on training in the field of software development, administration and information security for high-performance computing systems. This program is open to students with bachelor's degree in Information Security, Information Systems, Mathematics, Computer Science and Mechanics. They come to Tyumen State University not only from Russia but also from Kazakhstan, Tajikistan and other CIS countries. Although the admission requires a basic knowledge of computer science, the students have different levels of knowledge in algorithms and data structures, programming languages and technologies. Moreover, students have different motivations. Some of them try to improve the previously obtained knowledge and skills, while others want to explore new areas. That is why students can choose a specialization that extends their knowledge and skills obtained in their bachelor's degree (for example, programmers specialize in information security). Some students have just obtained their bachelor's degrees, while others have already been working in research and IT departments of large Russian and foreign companies (Rosneft, Gazprom, Schlumberger, Halliburton etc.); therefore, they have specific professional interests. At the same time, all students regard acquiring the knowledge and experience in the field of computer simulation to be very important. This is largely due to the fact that specialists in the field of design and development of applications for the study of geological objects, computational experiments to study physical processes in wells and bottomhole zone, visualization of hydrodynamic and geological models, processing and interpretation of seismic data, etc. are in high demand in the Tyumen region.

Since the master's program is related to the HPC area, students must learn to implement parallel and distributed applications for computational experiments with different models. The development of such programs inevitably involves the study of key problems of low-level parallel programming, such as load balancing and fault tolerance in computations.

This paper discusses the research project based methodology of teaching parallel programming to master's students in a High Performance Computing program. In addition to the classic tasks, students explored pairing algorithms, load balancing and fault tolerance through implementation in distributed applications and testing in computational experiments.

1.2 Related work

A lot of research has been dedicated to the issues in PDC education. However, at present there is no clear idea of the overall structure of the training and the content of undergraduate and graduate courses. In particular, many questions remain open: does the curriculum include one course or several, what is their volume, what initial training is needed, whether to teach it in introductory or advanced courses. At the same time, the fundamental courses in Computer Science dealing with languages, methods and technologies of programming, structures and algorithms of computer data processing, computer geometry and graphics, discrete and computational mathematics should take into account the current trends in the training of IT professionals. The content of these courses will be incomplete without highlighting the role and capabilities of algorithms and parallel programming technologies in software implementation of solutions of various applied problems. Graham [7] shows how parallel programming concepts (threads programming, system calls programming, virtual machine programming, message passing operations) might be introduced in CS curriculum. John & Thomas [2] present the distributed approach for PDC integration with traditional CS courses through the special modules and discuss the advantages of these courses providing students with an understanding of main PDC concepts and new experience in parallel programming. Brown & Shoop [8] argue for including flexible modules in CS courses and continuous teaching parallelism and concurrency at all undergraduate levels. Burtscher et al. [9] describe an innovative approach to introduce PDC concepts in the short modules taught across several lower division CS courses without an overhaul of the curriculum. However, independent courses in parallel programming that are geared towards the practical development of models and technologies of PDC (OpenMP, MPI, pthreads) for low and high divisions have become more widespread.

Pacheco [10] justifies the possibility of studying

PDC basics in lower division undergraduate courses and offers an introductory course for the undergraduate students with minimal background in CS [11]. Gergel & Kustikova [12] present undergraduate course "Introduction to Parallel Computing" for the 2nd year students with skills in the software development and basic mathematical knowledge. Stojanovic and Milovanovic [13] present undergraduate (8th semester) course designed to improve students' parallel programming skills. According to them, students get hands-on experience with programming for the shared and distributed memory computers in limited budget conditions. Yazici et al. [14] substantiate the effectiveness of the special teaching approach for undergraduate elective course "Parallel Computing" for computer engineering students. This methodology includes using real-life analogies to introduce main parallel concepts (pipeline, master-worker, SPMD), supplementing theoretical slides with the demonstration of real parallel code and small team projects. Wilkinson et al. [15] describe new pattern-based approach for teaching parallel programming with higher-level patterns in the special framework developed for the implementation and execution of the parallel and distributed programs. According to them, this approach has more advantages than disadvantages and main benefit is building the foundation for the future professional application development.

Although the NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing [1] does not require detailed study of many issues (data flow, load balancing, fault tolerance etc.), their place in graduate courses is not exactly defined. Differences in students' background and variety of master's programs further complicate the development of graduate courses in PDC. Muresano et al. [16] present active learning methodology for graduate course in parallel programming concentrated on teaching master-worker and SPMD concepts and developing efficient applications through students' experiences. According to them, active learning help the students to overcome such key PDC issues as mapping, scheduling, load balancing etc. Ponce et al. [17] discuss the need for masters' programs in HPC and note that in some cases students have to use non-academic options. Relying on their review of the academic and non-academic education programs, they prove the effectiveness of the design for master's programs in HPC, based on the experience in not-for-credit training in HPC at the SciNet HPC Consortium at the University of Toronto. With regards to the need of HPC and PDC courses that are available for students of various IT Programs, Wilson and Dey [18] describe four courses in scientific computation (starting with "Introduction to Scientific Programming" and concluding with "Distributed and Grid Computing for Scientists and Engineers"). They include hands-on exercises, labs and semester projects (both formal assignments and projects proposed by students), and substantive opportunities of the proposal methodology for the development of skills in applied parallel programming for upper division undergraduate and graduate students. Gonçalves et al. [19] substantiate the importance of OpenMP in-depth study in graduate course and demonstrate the connection between the fundamental knowledge of the parallel computing system architecture and the quality of the parallel program code.

The closest to our methodology are Wilson and Dey [18] in context of the research projects and Muresano et al [16] in the context of overcoming key PDC issues. However, we focus on the modeling and simulation of load balancing and fault tolerance, as well as different parallel programming languages and libraries.

1.3 PDC terminology

This paper uses terminology in parallel and distributed computing [1], some of the more commonly used terms are specified here.

Process—running program with resources (virtual memory addresses, access to the files and ports) for the execution of program instructions.

Thread—lightweight process. Each process has at least one thread. Thread has local data but shares the same virtual memory addresses and other resources of the parent process. Data transfer between the threads is supported by shared memory.

Shared memory—architecture of the computing system, based on the direct access of all the system's processors to the common physical memory.

Shared memory programming model—model, in which parallel processes (threads) have direct access to the same variables in the shared memory.

OpenMP (Open Multi-Processing)—open standard that supports shared memory programming model. OpenMP describes a set of compiler directives, library functions and environment variables that are intended for programming multi-threaded applications in C/C++ and Fortran.

Distributed memory—architecture of the computing system, where each processor can only address its own memory, but has network based access to the physical memory of all the system's processors.

Distributed memory programming model—model, where parallel processes of the distributed application have direct access to the local processor memory only and must use network communications to access memory on other processors where other processes are executing.

Message passing—type of communication between processes, where processes can send messages (data) to other processes and receive messages from them.

MPI (Message Passing Interface)—the standard for producing special libraries that supports the distributed memory programming model based on message passing. At the same time, MPI is a method of performing distributed computations. There are numerous message passing libraries (MPI distributions), that are intended for programming distributed applications in C/C++, Fortran, Python and other programming languages.

Synchronization—coordination of parallel processes (threads) in real time to ensure correct execution of algorithms, avoiding incorrect access to variables in shared memory (for threads), deadlocks, etc.

Load balancing (in the context of parallel programming)—special algorithm that improves the distribution of parallel processes between multiple computing nodes. Balancing can be performed both to optimize the load of the entire computing system, and to provide fault tolerance by reserving resources.

Fault tolerance (in the context of parallel programming)—property of the distributed application to continue execution in the event of failure of some computing nodes. Such stability may be provided by the application itself, system software, hardware or a combination of all of them.

Checkpoints—instant copy of the current state of the data that is used by the application. Checkpoints are important for fault tolerance because they allow restoration of a distributed application in the event of a node failure, continuing the execution of the corresponding process from the point of the last saved state, and not performing a full restart.

2. Course content

2.1 Place of the course in the curriculum

Individualization of training with the possibility of specialization in the field of software development, administration or protection of high-performance computing systems is the guiding principle of our master's program. To reach this goal, the program offers elective courses (one course per semester at the student's choice). However, all students are required to take courses in the basic disciplines (Table 1). This program allows a shift of emphasis and even the exclusion of certain topics in the elective courses. Here we take into account students' preferences and the subjects of their research or application projects. At the same time, for basic disciplines all topics are mandatory.

The course on "Algorithms and Parallel Programming" takes a special place in the curriculum due to its integrative approach [20]. The main goal of this course is fairly standard—to teach students to develop parallel algorithms and programs in following stages [21, 22]:

1. to select subtasks in the initial task that can be executed in parallel;
2. to adapt the initial task for managing the execution of subtasks;
3. to develop a parallel algorithm for subtasks and an algorithm for managing sub-tasks;
4. to choose parallel programming technology and implement algorithms;
5. to evaluate the effectiveness of the solution obtained in the course of computational experiments.

Ultimately, the course prepares students to develop software for high-performance computing

Table 1. Mathematics and IT courses (2016/2017 academic year)

Mandatory courses	**Elective courses**
First semester	
Algorithms and Parallel Programming	Information Security Methods
Modern Programming Languages	MATLAB for Simulation
Discrete Optimization	Communication Networks
Second semester	
Algorithms and Parallel Programming	Open Technologies for Software Development
Modern Programming Languages	Data Analysis
Computer System Architecture	Distributed Storage Systems
Administration and Security of Distributed Systems	
Third semester	
Simulation Systems	Optimal Control Problems
IT-Project Management	Mobile Applications Development
	Cloud and Distributed Computing Security

systems for real-world applications. At the same time, it is possible to single out a number of additional aims arising from the main goal. On one hand, such an integrative course makes it possible to assess the quality of the content and teaching of many disciplines that have been studied in a bachelor's degree. Throughout the teaching process, we observe how well students understand the basic concepts, such as computer architecture, operating systems, network technologies, language semantics and programming models, compilation theory, computer data structures and algorithms, models and methods of discrete and computational mathematics. In addition, practical tasks for this course are based on the content of both basic and elective courses that students take at the same time. Therefore, the quality of these assignments reflects quite well the degree to which a student understands new concepts. On the other hand, we can provide the opportunity to personalize the training depending on the level of basic knowledge and skills, as well as the preference for in-depth study of high or low-level programming, administration, and information security. This will allow us to react quickly to the arising difficulties due to insufficient initial preparation. Moreover, the focus on obtaining concrete results in the implementation of projects, carrying out computational experiments and working on a user interface helps students to use and rethink the content of those courses that they are taking at the same time (see Table 1). Thus, the additional objectives of the course "Algorithms and Parallel Programming" are to evaluate students' level of initial training in the field of computer science and monitor the achievements of students in general during the first year of studying.

2.2 Course content

The course "Algorithms and Parallel Programming" consists of 35 lectures and 35 practical sessions. We determined the content of lectures proceeding from the fact that the primary goal of this course is teaching parallel programming. The architecture of parallel computers and distributed systems, connection networks, special programming languages (Haskell, Erlang, Elixir), deployment and performance management applications are covered in detail in other courses. That is why lectures include overviews of basic information about the architecture for multiprocessor systems, models and technologies of parallel programming (Open MP, MPI, CUDA), and multithreaded programming (C#, C/C++).

The lectures use a small set of problems (solving a system of linear algebraic equations, numerical integration) to illustrate various PDC technologies. This helps students in the development of their own applications in practical classes, which have a completely traditional goal—the concretization and application of theoretical knowledge for solving applied problems.

During the practical sessions, students perform mandatory tasks of two types:

(1) a computational experiment to determine the execution time, speedup and efficiency of parallel algorithms, or to compare simultaneously running different decision algorithms of the same problem (numerical integration, search and sorting, graph algorithms, numerical solution systems of algebraic equations);
(2) applications' implementation for calculations on the given mathematical models and visualization solutions using multithreading for improved user interface (gradient methods for finding the extremum problem, numerical methods for solving boundary value problems for 2D heat transfer equation and Poisson's equation).

In the first case, students should use C/C ++ programming language and OpenMP, MPI, CUDA, OpenACC technologies. For assignments of the second type, students can choose any programming language and technology (although most students prefer the C # language in combination with Microsoft Task Parallel Library [23] and MPI.NET [24, 25]). This can be explained by the fact that C # .NET has become almost a must-have for Russian undergraduate programs. Throughout the course, as the algorithms and technologies of parallel programming are learned, tasks are increasingly aimed at creating applications for computer modeling. They include the requirements for data generation, controlled by a series of computational experiments and the collection of results by a separate control thread (process).

It is worth noting that we do not require the mandatory use of parallel programming patterns for performing these tasks [21], since this issue is studied in other courses (Modern Programming Languages, Computer System Architecture).

Our goal is for students to independently discover two important features of developing parallel and distributed applications:

(a) many parallel programming tasks use standard architectural solutions; this helps one to quickly develop an application prototype;
(b) however, standard solutions do not take into account hardware particularities, which is the crucial task of high performance computing. Therefore, the programmer should provide for the features of program execution on real equipment, that is, solve the problems of low-

level parallel programming (we use the term "low-level" not in the context of the programming language, but speaking about the degree of proximity of the task to the hardware level).

We do not limit the practical experience of students in compulsory assignments. In addition, the course includes individual projects in the 2nd semester. Depending on their specialization (software development, administration of computer systems or information protection), a student can select an application task to develop a parallel or distributed application. It is during the implementation of these projects that students explore the issues of low-level parallel programming—load balancing and fault tolerance by the application itself.

3. Proposed methodology

Practical tasks on parallel programming are focused on the software implementation of a fairly standard set of algorithms for solving problems of discrete and computational mathematics. However, this level of training is not sufficient to carry out research at a master's level and implement applied projects using distributed computing technologies. Creating computer simulations based on high-performance computing is impossible without the experience of solving such issues of low-level parallel programming as the data flow management, synchronization, load balancing and fault tolerance. Any programmer who develops software for distributed systems encounters these issues. Of course, experts in the field of administration and protection of distributed computing systems should also understand the essence of the corresponding problems and have an idea of the methods for solving them. We believe that the best way to solve these problems is to compute the appropriate algorithms in the application, and then carry out computational experiments. Therefore, as a main tool for the practical study, we offer the implementation of special project tasks. The main problem that had to be solved when creating tasks was how to simultaneously (a) provide a single logic for setting tasks; (b) take into account the specialization of students; (c) allow students to determine the level of the solution to the original problem, but not below the minimum requirements.

Tasks include exploring data distribution (mapping), synchronization, load balancing and fault tolerance. Projects vary in level of difficulty and the formalization or, on the contrary, the openness of the original problem statement. At the same time, all tasks are based on generalized sector models, which ensures a common base and achievement of learning objectives. We use the term "generalized" here because the problems can include domain decomposition, pre-existing connected or overlapping areas, and data decomposition. At the same time, we do not limit the application of such models to computer simulations of physical processes only (such as sector hydrodynamic reservoir model). While we offer all students a unified statement of the general problem, it can have different content.

While developing the course tasks, we have used not only our teaching experience of parallel programming for undergraduate and graduate students. We also relied on the existing practice of the development of distributed computing systems. In particular, the Department of Software and Information Security carried out a project to develop a distributed system for hydrodynamic modeling based on sector models for both undergraduate and graduate students. It was important that it was the students who sought to try different solutions. They tested parallel algorithms for coupling sector models, compared the technologies for implementing distributed computing, developed heuristic algorithms for dynamic load balancing and so on. At the same time, we first offered the students a rather abstract statement of the problem. Only after the prototype of the system was created, we supplemented the original problem with the physical and geometric features of model interfacing, the requirements for synchronization and data balancing. In this way, a bridge was naturally constructed from the use of manual mapping to runtime load balancing, from handling exceptions to ensuring the fault tolerance of the entire system.

Our experience has shown that the presented approach to development based on the generalized application of sector models allowed to proceed gradually from educational problems to the implementation of full-scale research projects.

The students were expected to explore a set of issues for solving the following problems:

- The rules and interface conditions (distribution/ synchronization) of models (solutions in the subdomains).
- Load balancing on the compute nodes.
- Providing fault tolerance during data processing.

Solutions for the problem of balancing and fault tolerance in educational applications can be quite formal, without regard to the nature of the original problem. However, the first problem requires special attention: specifically, features of interface conditions can be decisive for the choice of methods of load balancing and fault tolerance. For example, in one of the lectures we showed students that when solving the Dirichlet problem by the domain decomposition method [26], it may be necessary to change the grid in one of the subregions due to the peculiarities of the boundary conditions. This would entail a change in both the interface conditions, and the distribution of the load on the nodes. Then we added the fault tolerance requirement: we needed to get a solution in case of failure of one of the computing nodes. For this, it was necessary to take into account all these features in the model prototype of the future program and to perform computational experiments for choosing the optimal solution. We used similar examples in other lectures to show the need for computer modeling not only to solve the original problem, but also to develop a parallel or distributed application.

For our proposed course methodology, we take into account both the learning objectives of the course and various professional interests of students. Therefore, we offer students two types of interface conditions problem. The first option is a formal statement of the problem (the transformation of arrays, lists, trees) and interface conditions determined by formal rules (e.g., the calculation of the generalized aggregating function, merge sort etc.). The second option involves solving applied problems. Accordingly, the applied problem (heat and mass transfer, optimal route planning, search in the attack graph etc.) and interface conditions are based on specific subject areas (e.g., physical laws, connections in social networks, regulatory criteria in the search for vulnerabilities in distributed systems and so on).

The requirements for the load balancing and fault tolerance were set as follows (we have given optional conditions in parentheses):

Load balancing on the nodes is determined in advance (run-time) based on the characteristics of the volume of raw data, the computational complexity of the algorithm and performance of the nodes (including fault history).

The history of failures for the last month is stored in a file (txt, csv, xslx, xml). The student must evaluate the distribution of the failure rate and propose solutions within the acceptable probability of failure. The failure history can be analyzed using additional tools, for example, Python, R, MATLAB libraries. But it is more preferable that the analysis and visualization tools are included into the functions of the program.

One of following methods should provide fault tolerance:

- storages for checkpoints located on the local disks of nodes (the central node);
- local intermediate results periodically (depending on the expected failure rate) are sent to the other nodes (only central node);
- restart all the processes (in case of failure of any node, data is re-sent from the central node).

In all cases, we assume the absence of central node failures. The student must determine the control points him- or herself in accordance with the parallel algorithm and the program architecture and assess how the chosen method of fault tolerance affects the execution time of the program. We do not limit students in the choice of equipment for developing and deploying applications. They can use a training cluster, 4-core computers and a local network in a training laboratory or home network from any suitable devices. Simulation of distributed computing on one node, as well as a failure simulation using a request to cancel a task, is also permissible.

Students work on the project in the 2nd semester for 12 weeks (see Table 2). The lecturer evaluates the quality of the project at all stages. In addition, the entire group of students participates in the evaluation of the user interface and project presentation. Accordingly, the total score determines the final grade: A (90–100), B (76–89), C (61–75), D (0–60).

4. Evaluation of proposed methodology

The course "Algorithms and Parallel Programming" for graduate students has been taught for 4 years (2013–2016). A total of 73 students attended it, out of which 58 students completed their projects in accordance with the plan (see Table 3).

The results of the projects in the 2013/2014 academic year showed a few problems with the first version of the plan, which is described in Table 4.

Therefore, we made the project plan more detailed (see Table 2) and used the requirements of stages 1–4 of the new plan also for routine practical tasks. This allowed us to improve the course content and the methodology, as well as gain valuable experience. We significantly changed the time allocated for the implementation of the sequential algorithm for solving the basic problem. Initially, we had allocated a period of 4 weeks for this task, and for this reason, many students did not show sufficient activity. Furthermore, experience has shown that students do not pay enough attention to the design of the user interface. At the same time, the user interface is very important for computer modeling and simulation systems. In addition, the quality of the interface reflects the conscientiousness of the student and the professional attitude to the development of programs. Therefore, interface development has become a separate stage of the project. Finally, we decided to allocate a different number of maximum points for each stage in order to show students the relative importance of each project stage.

We applied the proposed methodology in its final form in 2015–2016 for a course with 37 students.

Table 2. Project schedule

Stage	Week	Results at this stage	Score (max)	What is being assessed ?
1	1–2	Implementation of sequential algorithms for solving the basic problem in the console application	10	Code quality including accuracy of algorithm implementation, reliability, efficiency, data structures, style, comments (source code has to be submitted to the lecturer and instructor)
2	3–4	Development and implementation of the data (or domain) decomposition algorithms and interface conditions of particular solutions	15	
3	5–6	Development and debugging of the distributed application (MPI)	15	Application quality including efficiency and scalability (source code, the description of the development and debugging stages have to be submitted to the lecturer and instructor)
4	7	Design of the user interface	10	Usability including learnability and efficiency (The lecturer and students evaluate the demo version of the corresponding module).
5	8	Development of balancing algorithms and their inclusion in the main application	15	Application quality including accuracy of balancing (fault tolerance) algorithm implementation, efficiency and scalability (source code and the description of the additional algorithm have to be submitted to the lecturer and instructor)
6	9–10	Developing fault tolerance algorithms and their inclusion in the main application	15	
7	11	The computational experiment and analysis of results	10	The quality of the computational experiment and the validity of the results (description of the computational experiment design and data analysis methods, the results in the form of tables, charts and graphs have to be submitted to the lecturer)
8	12	Project presentation	10	Presentation skills (The lecturer and students evaluate the structure and content of the report, the clarity of the presentation, the answers to the questions and communication with the audience)

Table 3. Projects' results: the distribution of final grades

	Academic year			
Final evaluation	2013/2014	2014/2015	2015/2016	2016/2017
A	0	2	2	4
B	3	3	5	7
C	9	8	6	9
D	5	6	1	3
Total number of students	17	19	14	23

Table 4. Project schedule (2013/2014)

Stage	Week	Results at this stage	Score (max)
1	1–4	Implementation of sequential algorithms for solving the basic problem in the console application	10
2	5–8	Development and implementation the distributed application (MPI) in accordance with the decomposition algorithms and conjugation of particular solutions	30
3	9	Development balancing algorithms and their inclusion in the main application	20
4	10	Developing fault tolerance algorithms and their inclusion in the main application	20
5	11	The computational experiment and analysis of results	10
6	12	Project presentation	10

Table 5. Intermediate results of assignments

		Scores' statistics					
		Applied tasks (8 students)			Formal tasks (29 students)		
Stage	**Score (max)**	**max**	**min**	**median**	**max**	**min**	**median**
1	10	1.00	0.70	0.90	1.00	0.70	0.80
2	15	0.93	0.53	0.80	0.87	0.40	0.73
3	15	0.87	0.53	0.80	0.87	0.40	0.67
4	10	1.00	0.70	0.80	1.00	0.80	0.90
5	15	0.93	0.40	0.73	0.80	0.33	0.60
6	15	0.87	0.33	0.80	0.87	0.33	0.53
7	10	1.00	0.60	0.80	1.00	0.50	0.70
8	10	1.00	0.70	0.90	1.00	0.70	0.80
Total	100	0.93	0.55	0.79	0.90	0.51	0.68

Table 5 shows the combined relative characteristics of students' grades, specifically the maximum, minimum values and median with respect to the highest possible score—Score (max).

A small number of students—8 (21.6%) chose applied tasks for their projects, but out of these only five students (62.5%) were able to complete their projects (in other words, they received a score of no less than 75% at each stage). 29 students (78.4%) have chosen a formal task, 13 of them (44.8%) were also able to complete their projects. The students in the first group achieved higher results, although the level of complexity of applied and formal assignments was approximately the same. We assumed that students who fulfilled projects corresponding to their specialization had the necessary motivation. Conversations with students confirmed this assumption. We have analyzed how the students carried out various parts of the project. We found that simulation of random failures and fault tolerance were the biggest challenges for students. However, 22 students (59.5%) were able to implement the simple solution (restart all processes).

The detailed assessment of the project's quality at all stages (see Table 2) revealed the results of taking different courses in the bachelor's program. The choice of programming languages and technologies confirmed the popularity of .NET technologies: 20 students (54.1%) used C # (MPI.NET), 2 students (5.4%)—C # (Windows Communication Foundation), 11 students (29.7%)—C ++ (MPI), 3 students (8.1%)—C ++ (OpenMP + MPI), 1 student (2.7%)—Erlang. The overwhelming majority of students (89.2%) preferred Windows OS, which they previously studied in the course on Operating Systems. However, almost all these students needed the help of an instructor to debug a parallel program, because they were only superficially familiar with the work of threads at the OS level. At the same time, students who used Linux OS were more creative. They previously studied the features of debugging programs independently and successfully coped with this task. The analysis of the source code showed that all students who completed projects competently designed the application as a whole, preferring the object oriented programming model. They freely implemented standard graph algorithms and other methods of discrete mathematics, but they needed help to formalize the applied problem. Insufficient training in the field of mathematical modeling was the reason for refusing to solve applied problems. Nevertheless, students successfully coped with computational experiments, which they repeatedly carried out when solving simple tasks on parallel programming (multiplication of matrices, solving systems of equations, etc.). The results showed that the students performed computational experiments and analyzed the data to a very high standard. At the same time, 24 students (64.9%) included the necessary tools into the program and implemented a computer simulation system. However, the students noted that the material of mathematical courses could contain more tasks related to the modeling of parallel and distributed computing. In addition, the students evaluated the course organization and their results through the questionnaires. Regardless of their final grade, the majority of students – 31 (83.8%) noted the importance of such projects for the practical study of the problems associated with the development of distributed systems.

5. Conclusion

This paper has presented a methodology for preparing graduate students to implement application projects that use distributed computing technologies. The goal of this method of teaching is to enable students to gain practical experience in solving problems of low-level parallel programming. Our approach allows us to build a complete system of project-type tasks with different levels of complexity: from connecting formally processed arrays or images to interface conditions of sector hydrodynamic models. In addition to the classic tasks for parallel programming, they include the study of algorithms and rules of interface conditions, issues of load balancing and fault tolerance.

Resolving the issues of data analysis and computer modeling is also an obligatory part of the projects. Our experience has shown that such projects are quite complex to implement in full. However, they help students to identify the key problems of the computational systems development based on high performance computing technologies. These issues are not usually included in traditional parallel programming tasks, although they encourage students to apply the material of various CS courses (Math Modeling, Simulation, Graph Theory, Data Analysis etc.) to solve a particular problem. We understand that complex assignments for all graduate courses would be an ideal tool for the professional development of students. Evaluation of the results of using the proposed methodology prompted the authors to start developing practical cases: "Modeling and simulation of distributed application failures" for the basic course "Simulation Systems"; "Static and dynamic load balancing for a distributed computing system" for the elective course "Optimal Control Problems"; "Attack Graph Generation and Visualization" for the elective course "Cloud and Distributed Computing Security". We hope that the methodology presented in this paper will be useful and inspire lecturers to go beyond their standard course curriculum and use similar projects both for developing programming skills and studying PDC concepts, as well as motivating students to use methods of applied mathematics and computer modeling for the development of computer systems.

References

1. S. K. Prasad et al., NSF/IEEE-TCPP Curriculum Initiative on Parallel and Distributed Computing—Core Topics for Undergraduates, Version I, Dec 2012, https://grid.cs.gsu.edu/~tcpp/curriculum/sites/default/files/NSF-TCPP-curriculum-version1.pdf, Accessed 06 October 2017.
2. D. J. John and S. J. Thomas, Parallel and Distributed Computing across the Computer Science Curriculum, *IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW)*, 2014, Phoenix, AZ, USA, 19–23 May 2014, pp. 1085–1090.
3. V. Voevodin, V. Gergel and N. Popova, Challenges of a Systematic Approach to Parallel Computing and Supercomputing Education, *Parallel Processing Workshops (Euro-Par 2015)*, 2015, Vienna, Austria, 24–28 August 2015, pp. 90–101.
4. V. Gergel, A. Liniov, I. Meyerov and A. Sysoyev, NSF/IEEE-TCPP Curriculum Implementation at the State University of Nizhni Novgorod, *IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW)*, 2014, Phoenix, Arizona, USA, 19–23 May 2014, pp. 1079–1084.
5. Professional Standards "Communication, Information and Communication Technologies", http://fgosvo.ru/docs/101/69/2/6, Accessed 06 October 2017.
6. Professional Standard "System Programmer", http://fgosvo.ru/uploadfiles/profstandart/06.028.pdf, Accessed 6 October 2017.
7. J. R. Graham, Integrating Parallel Programming Techniques into Traditional Computer Science Curricula, *SIGCSE Bull.*, **39**(4), 2007, pp. 75–78.
8. R. Brown and E. Shoop, Modules in Community: Injecting More Parallelism into Computer Science Curricula, *The 42nd ACM Technical Symposium on Computer Science Education (SIGCSE 2011)*, 2011, Dallas, Texas, USA, 9–12 March 2011, pp. 447–452).
9. M. Burtscher, W. Peng, A. Qasem, H. Shi, D. Tamir and H. Thiry, A Module-based Approach to Adopting the 2013 ACM Curricular Recommendations on Parallel Computing, *The 46th ACM Technical Symposium on Computer Science Education (SIGCSE2015)*, 2015, Kansas City, Missouri, 4–7 March 2015, pp. 36–41.
10. P. Pacheco, Teaching Parallel Programming to Lower Division Undergraduates, *The 1st NSF/TCPP Workshop on Parallel and Distributed Computing Education (EduPar 2011)*, 2011, Anchorage, Alaska, USA, 16 May 2011.
11. P. Pacheco, *An introduction to parallel programming*, Elsevier, 2011.
12. V. Gergel and V. Kustikova, Internet-Oriented Educational Course "Introduction to Parallel Computing": A Simple Way to Start, in V. Voevodin and S. Sobolev (eds), *Supercomputing. RuSCDays 2016. Communications in Computer and Information Science*, **687**. Springer, Cham, 2016, pp. 291–303.
13. N. Stojanovic and E. Milovanovic, Teaching Introductory Parallel Computing Course with Hands-On Experience, *International Journal of Engineering Education*, **31**(5), 2015, pp. 1343–1351.
14. A. Yazici, A. Mishra and Z. Karakaya, Teaching Parallel Computing Concepts Using Real-Life Applications, *International Journal of Engineering Education*, **32**(2), 2016, pp. 772–781.
15. B. Wilkinson, J Villalobos and C. Ferner, Pattern Programming Approach for Teaching Parallel and Distributed Computing, *The 44th ACM Technical Symposium on Computer Science Education (SIGCSE2013)*, 2013, Denver, Colorado, USA, 6–9 March 2013, pp. 409–414.
16. R. Muresano, D. Rexachs and E. Luque, Learning Parallel Programming: a Challenge for University Students, *Procedia Computer Science*, **1**(1), 2010, pp. 875–883.
17. M. Ponce, E. Spence, D. Gruner and R. van Zon, Scientific Computing, High-Performance Computing and Data Science in Higher Education, https://arxiv.org/pdf/1604.05676.pdf, Accessed 06 October 2017.
18. L. A. Wilson and S. C. Dey, (2016, November). Computational Science Education Focused on Future Domain Scientists, *Workshop on Education for High Performance Computing (EduHPC 16)*, 2016, Salt Lake City, Utah, USA, 13–18 November 2016, pp. 19–24.
19. R. Gonçalves, M. Amaris, T. Okada, P. Bruel and A. Goldman, OpenMP is Not as Easy as It Appears, *The 49th IEEE Hawaii International Conference on System Sciences (HICSS)*, 2016, Koloa, Hawaii, USA, 5–8 January 2016, pp. 5742–5751.
20. I. Zakharova and A. Zakharov. Integrative Possibilities of the Course "Parallel Programming", *The 1st Russian Conference on Supercomputing Days (RuSCDays 2015)*, 2015, Moscow, Russian Federation, 28–29 September 2015, pp. 759–762.
21. T. G. Mattson, B. Sanders and B. Massingill, *Patterns for Parallel Programming*, Pearson Education, 2004.
22. M. McCool, J. Reinders and A. Robison, *Structured Parallel Programming: Patterns for Efficient Computation*, Elsevier, 2012.
23. D. Leijen, W. Schulte and S. Burckhardt, The Design of a Task Parallel Library, *ACM SIGPLAN Notices*, **44**(10), 2009, pp. 227–242.
24. D. Gregor and A. Lumsdaine, Design and Implementation of a High Performance MPI for C# and the Common Language Infrastructure, *The 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, 2008, Salt Lake City, Utah, USA, 20-23 February 2008, pp. 133–142.
25. M. Hafeez, S. Asghar, U. A. Malik, A. ur Rehman and N.

Riaz, Survey of MPI Implementations, *International Conference on Digital Information and Communication Technology and Its Applications (DICTAP 2011)*, 2011, Dijon, France, 21–23 June 2011, pp. 206–220.

26. V. Dolea, P. Jolivet and F. Nataf, *An Introduction To Domain Decomposition Methods: Algorithms, Theory and Parallel Implementation*, https://hal.archieves-ouvertes.fr/cel-01100932/document, Accessed 06 October 2017.

Irina Zakharova is a Professor and the Head of the Software Department at University of Tyumen, Russian Federation. Her research interests include Math Modeling, Simulation, Parallel Programming, Open Source Math Software, High Performance Computing Education, IT in Education. She has published more than 100 scientific articles, 7 textbooks and 4 book chapters in these fields. She has a vast experience in distance education related to Programming Languages courses.

Alexander Zakharov is a Professor and the Head of the Information Security Department at University of Tyumen, Russian Federation. His areas of research and interest are Information Security of Distributed Systems, Cloud Computing, Medical Information Systems, Data Analysis, Internet of Things, and Computer Networks. He has published many scientific articles and research reports in the field of Information Security and Medical Information Systems. Prof. Zakharov is the Head of Tyumen Cisco Networking Academy and has a large experience in online education covering courses in Information Security of Distributed Systems and Computer Networks.