Teaching Practice about Parallel Scientific Computing for Graduate Students Majored in Physics

Sanhuang Ke\textsuperscript{a,}\textsuperscript{*} and Yuzhong Zhang\textsuperscript{b}

School of Physics Science and Engineering, Tongji University, 1239 Siping Road, Shanghai 200092, P.R. of China

\textsuperscript{a}shke@tongji.edu.cn, \textsuperscript{b}yzzhang@tongji.edu.cn

*Corresponding author

Keywords: Teaching practice, Parallel computing, Graduate student, Physics.

Abstract. With the rapid developments of computer hardware and software, nowadays parallel computing is playing more and more important roles in the scientific research in physics related fields. The broad use of supercomputers and computer clusters makes it possible to solve much more efficiently various scientific problems by computations and simulations. Therefore, to gain basic knowledge and programming skills of parallel computing is very necessary for junior graduate students majored in physics, while the teaching of these knowledge and skills has been lacking in the traditional course of computational physics. Here, we present a teaching practice of parallel computing for junior graduate students in physics, which has been shown to be successful by the homework presentation of the students.

Introduction

In recent years, parallel computing has become a more and more important tool used in the scientific research of physics and other related fields. This is due to the rapid developments of computer hardware and software, though the exponential growth of Moore’s law is slowing down. Currently, server clusters of different sizes have been broadly used in many research groups of universities in China. Also, more and more supercomputers located in different cities are becoming accessible to research groups of universities. The rapid growth of the parallel computing resources is providing powerful tools for solving various scientific problems in physics related research projects, including projects in theoretical physics, condensed matter physics, and materials science, etc. However, in traditional courses of computer language for undergraduate students and of computational physics for graduate students, the systematic teaching of parallel computing is still lacking in universities of China. This situation is mismatching the current scientific research progress and may cause a big waste of these resources. Therefore, to set up a course about parallel computing becomes very necessary for junior graduate students in physics, so that they can gain systematically the basic knowledge and programming skills and can be prepared and ready to deal with the research work when they are involved in their research projects soon. Otherwise, they may have to learn the related knowledge by themselves assuming more time and with more difficulties. However, to design such a course is not so easy. This is because parallel computing is associated with many subfields, including knowledge about computer hardware, operating system, programming language, and parallelization interface, etc. Each of these subfields may be devoted by an independent course. To let them to study fully all these courses is not allowed in terms of time and also unnecessary for graduate students in physics. This is because what they really need is just the ability and skill to solve some specific scientific problems without the need of knowing in depth the full subfields. In this paper, we present a course design which has been successfully applied to our teaching practice for junior graduate students in our school. In the following sections we give the logic of the course design and our conclusion.
Course Design

As mentioned above, parallel computing is associated with many different subfields, including the basic hardware knowledge (server motherboard, memory, CPU and GPU, hard disk array, etc.), operating system (mostly, Linux systems for server clusters and supercomputers), programming languages (mostly, C/C++ and Fortran in scientific computing), and parallelization interfaces (mostly, MPI for CPU parallelization and CUDA for GPU parallelization). The full learning of all these knowledge is not quite feasible and also unnecessary for graduate students in physics. What they really need is the practical use of the existing parallel code with possible minor/moderate modifications, or to program a specific code to solve the problems they meet in their research project. Therefore, the designed course should only cover, on one hand, the brief picture and the basic concepts of parallel computing and the related hardware, but has to cover, on the other hand, the detailed exercisable programming knowledge. This means that we need to do some tailoring for the content. In the followings, we present the major content of our course and the logic of the choices.

**Basic knowledge about the hardware.** Parallel computing is based on a corresponding hardware architecture which is very technical and complicated. Therefore, this part of content should not be taught in much details. The goal is to let the students to understand how a computer system or a computer cluster system works, including the roles played by the motherboard, CPU, memory, hard drive, and GPU, as well as by the network switch. The focuses are 1) the difference between the shared memory architecture and the distributed memory architecture, 2) the different features of CPU vs. GPU in scientific computing, and 3) how a computer cluster is built from individual computers via a network switch. As examples, several supercomputer systems [1] and a server cluster are described. In our practice, this part takes 4 class hours.

**Operating system.** Operating system is the interface between users and the hardware. Nowadays, graduate students are already very familiar with the use of the Windows operating system. However, most supercomputers and server clusters for scientific computing run Linux operating systems, such as Suse Linux enterprise, CentOS [2], etc, instead of Windows. Therefore, it is necessary for the students to gain a basic knowledge about the usage of Linux system, including the login/logout, shell, system environment, basic structure of the system directory, commonly used Linux commands, file management, process management, software installation, etc. As an exercise for homework, the students are requested to install a Linux system on their own Windows laptop computers, either by doing a dual-boot installation or by doing a virtual installation based on virtual machine softwares, such as VMWare. In this way, the students can practise all the commonly used Linux commands and become familiar with the usage of Linux operating system. In our practice, this part takes 4 class hours.

**Programming language.** Currently, most graduate students of China already learned the C/C++ language in their college. However, most open-source and commercial softwares they will use in their research are still programmed in Fortran (formula translator) language [3, 4]. For instance, almost all softwares for electronic structure calculations in condensed matter physics (i.e., Siesta, Quantum Espresso, VASP, Abinit, Gauss, etc.) are in Fortran90. Besides the historical reason, a more realistic reason for this is that compared to C/C++ Fortran still has some obvious advantages for scientific computing, especially, for parallel scientific computing, such as the direct matrix operations and the higher efficiency for multifold loop operations. Additionally, Fortran is very close to the nature language and is easy to learn for the graduate students who already know well C/C++. After considering all of these, we devote 4 class hours to teach the students Fortran90 programming. Obviously, this can only be the basics. If they need to do complicated programming in the future they can go further by looking up the reference books we give. As an exercise for homework, the students are requested to install a Fortran compiler, either open-source gfortran compiler or Intel Fortran compiler (academic version), on their Linux system.

**Parallelization interface: MPI (Message Passing Interface).** To date, MPI has become the most commonly used parallelization interface for CPU-based parallel scientific computing. MPI is a
universal interface and can be used for both the shared memory and distributed memory systems. It is convenient to use it to program complicated codes which can run on either a single multi-core workstation/PC or a top-level supercomputer. Thus, most open-source and commercial softwares involved in scientific research in physics, especially in condensed matter physics, are using MPI as the parallelization interface. MPI has more than 200 interface functions which can be invoked in different programming languages, such as Fortran, C, C++, Python, etc. However, by using only 10 (4 for setting up the parallel environment and the other 6 for communications) of these functions people can realize almost all parallelization functionality of their code. In our practice, we focus on teaching the syntax rules and functionalities of these 10 MPI functions, step by step. Then we take real and complex matrix multiplications as examples to show how these functions work. Different schemes for the parallelization and the efficiency considerations are also emphasized. As an exercise for homework, the students are requested to install the OpenMPI package [5] bound to their Fortran and C compilers on their Linux system, and to compile and run the taught example programs. This part takes our 6 class hours.

Parallelization interface: CUDA (Compute Unified Device Architecture). In recent years, general purpose GPU (GPGPU) computing plays more and more important roles in many fields, including artificial intelligence (AI) and scientific computing, which can provide even much more floating point computing power than CPUs. This is due to the rapid growth of the computing power of individual GPU and also the development of GPGPU programming environment, such as Nvidia CUDA. Although OpenCL may be more adaptive for heterogeneous computing, currently CUDA is more user friendly and more commonly used for GPGPU computing based on Nvidia GPUs. Unlike CPU, a GPU has much more computing cores, which is suitable the most for performing single instruction multiple data (SIMD) computations. GPU parallelization is usually combined with MPI parallelization, with one or several MPI processes communicating with one GPU device. In our practice, we focus on teaching the syntax rules and usage of the CUDA instructions [6,7]. By taking the previously mentioned MPI code of matrix multiplication as the prototype, we show the students how to add the CUDA environment and how to invoke CUDA instructions to perform a GPU accelerated matrix multiplication. As an exercise for homework, the students are requested to install the CUDA Toolkit package on their Linux system and to compile and run the taught example programs. This part takes our 6 class hours.

Programming practice. By dividing the students into groups, we ask each group to choose a specific problem in terms of their research projects and interest, which has large amount of data to compute and process. They are requested to find out the result or to solve the problem by doing their parallel computing programming, and finally, to present their work in a talk. This part will take about 2 class hours.

Summary

In summary, we have presented a teaching practice about parallel scientific computing for the graduate students majored in physics in our school. By designing properly the content of the course, we show that, in terms of their practical need, the learning of the basic knowledge and programming ability can be achieved in about only 26 class hours. The final presentation of their work indicates that this scheme is pretty successful.

This study is supported by Postgraduate Education Reform Project of Tongji University (Grant No. 2018GH103).

References

[1] Information on https://www.top500.org/
[2] Information on https://www.centos.org/
[3] J.C. Adams, W.S. Brainerd, J.T. Martin, B.T. Smith, and J.L. Wagener, Fortran 90 Handbook, Intertext Publications, McGraw-Hill Book Company, ISBN 0-07-000406-4, 1992.

[4] I. Chivers and J. Sleightholme, Introduction to Programming with Fortran, Springer. ISBN 1846280532, 2005.

[5] Information on https://www.open-mpi.org/

[6] Information on https://devblogs.nvidia.com/even-easier-introduction-cuda/

[7] G. Ruetsch and M. Fatica, CUDA Fortran for Scientists and Engineers: Best practices for efficient CUDA Fortran programming, Elsevier Inc., ISBN 978-0-12-416970-8, 2014.