The Ministry of Education and Science of the Russian Federation

Lobachevsky State University of Nizhni Novgorod

Computing Mathematics and Cybernetics faculty

The competitiveness enhancement program
of the Lobachevsky State University of Nizhni Novgorod
among the world's research and education centers

Strategic initiative
“Achieving leading positions in the field of supercomputer technology
and high-performance computing”

Introduction to MPI

Lecture 1. Introduction

Nizhni Novgorod

2014

Lecture_1_. Introduction

Among numerous laws that govern the computer world there is one that is truly fundamental: this is continuous hardware performance improvement. The importance of hardware performance growth is to a large extent due to industry demands as better performing computers facilitate solution of computationally intensive sci-tech problems. Furhermore, performance improvement makes it possible to solve more complex problems and extend the research frontiers on an ongoing basis. The complexity of problems to be solved by computers is now immense, being many times higher that we could imagine five or ten years ago.

Computer performance growth is also encouraged by continuous improvement in the field of hardware engineering. Hardware manufacturers have to comply with Moore’s law which states that the system performance must double each 18 months. Until quite recently, computer performance was to a large extent ensured by increasing CPU frequency. However, the possibilities offered by this approach are limited: at a certain point, further CPU frequency improvement will require considerable engineering effort thus entailing increased energy use and unsurmountable problems of heat control.

In this context, fundamental changes in the hardware engineering were inevitable, so a new general line was adopted thus making complex “solid” CPUs give place to “compound” CPUs consisting of relatively simple multiple peer cores. The maximum CPU performance is in this case equal to the total performance of its cores. Thus, packing CPUs with more and more cores, one can ensure performance growth without troublesome frequency growth.

This is the way the multicore era has put an end to the frequency race. Quad-core and octo-core processors have now become common, while hardware manufacturers claim that 12 to 16-core processors are on their way. Processors with hundreds and thousand cores are just around the corner.

However, one must understand that the adoption of multicore processors means transition to parallel computing. Indeed, to benefit from the advantages of multicore processors, one has to identify data-independent parts of computation processes and make each of them run on different cores. Such an approach helps reduce time expenditures while attainable speedup is limited only by the number of CPU cores and independent parts of the computation processes. Parallel computations become inevitable and omnipresent.

However, parallelism makes the efficient use of multicore systems more complicated. Parallel computing requires parallel generalization of the traditional sequential pattern of problem solving. Thus, for multicore systems, numerical methods should be developed as systems of parallel interacting processes allowing for running on independent cores. The applicable algorithmic languages and systemware must enable development of parallel programs and ensure synchronization and mutual exclusion of asynchronous processes etc. This course is dedicated to MPI, a parallel software development technology based on interaction of several processes.

All those parallelism-related problems aggravate the gap between computing potential of the contemporary systems and the existing algorithms and software to solve complex problems. Therefore, bridging this gap is one of today’s top priority sci-tech challenges.

The course includes 12 lectures:

Lecture 1. Introduction The lecture emphasizes the importance of parallel computations and describes the general course structure.

Lecture 2. Basic Notions and Definitions The lecture introduces the notion of parallel computations. It describes basic efficiency parameters and demonstrates applicability of such parameters by the example of the number sequence summation problem.

Lecture 3. MPI-Based Parallel Programming. Basic operations. The lecture gives basic notions and definitions related to MPI. It also lists the minimum set of functions required for parallel program development.

Lecture 4. MPI-Based Parallel Programming. Collective operations. The lecture describes an extended set of operations ensuring more efficient data exchange between computation processes. It also gives an application example for collective operations.

Lecture 5. Principles of Parallel Method Development. The lecture describes parallel method development stages and gives development examples.

Lectures 6-7. Parallel Methods for Matrix-Vector Multiplication The lecture is dedicated to the basic parallelization methods to solve the matrix-vector multiplication problem.

Lectures 8-9. Parallel Methods for Matrix Multiplication The lecture is dedicated to the basic parallelization methods to solve the matrix multiplication problem.

Lecture 10. Parallel Computation for Systems with Distributed Memory. This lecture describes a classification of computer systems and introduces the notion of parallelism for distributed memory systems.

Lecture 11. Parallel Computation Modeling and Analysis. The lecture is dedicated to the basic theory of parallel computation modeling and analysis. It introduces the notion of “operations-operands” graph and describes a number ways to estimate parallel method efficiency. Parallel computation modeling and analysis is illustrated by π computation and the finite difference method.

Lecture 12. Estimation of Communication Complexity for Parallel Algorithms. The lecture is dedicated to the issues of interprocessor communication time estimation. It compares the runtimes for models and experiments.