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 PARALLEL PROGRAMMING

Lecture 4. Principles of Parallel Method Development

Nizhni Novgorod

2014

Lecture _4_. Principles of Parallel Method Development

Development of parallel computation methods for solving time-consuming problems is always a serious work. To simplify the theme under consideration, we will leave aside the mathematical aspect of development and the proof of algorithm convergence, as these issues are to this or that extent considered in a number of "classical" courses of mathematics. Here we will assume that the computation schemes for solving the problems discussed further are already known.[1)] With regard to these assumptions, the course of actions to develop efficient parallel computation methods may be as follows:

- To analyze the available computational schemes and subdivide them (*decompose*) in parts (*subtasks*), which may be computed to substantial degree independently,
- To evolve the information interactions that should be carried out between subtasks in the course of solving the originally formulated problem,
- To define the computer system, which is necessary (or available) for solving the problem, and distribute the formulated set of subtasks among the system processors.

If we consider the problem in the most general way, it becomes evident that the amount of computations for each processor being used should be approximately the same. It provides for the uniform computational processor loading (*load balancing*). It is also evident that the distribution of subtasks among the processor should be carried out in such a way that the number of information links (*communication interactions*) among the subtasks should be minimal.

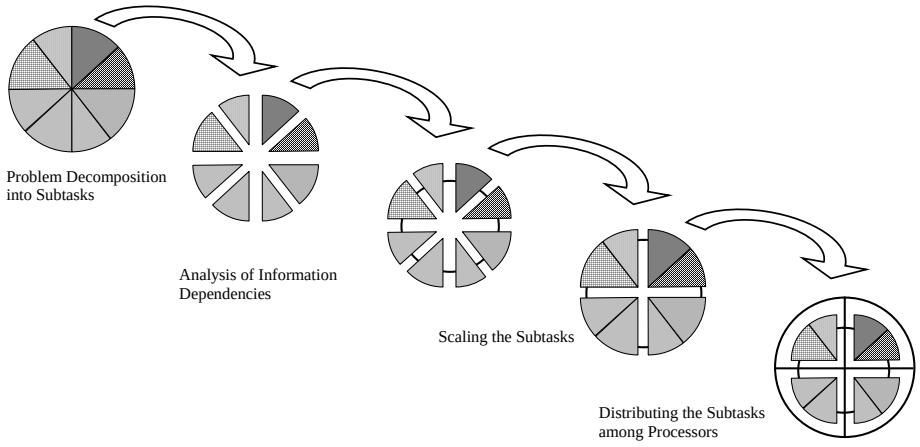

Figure 4.1. Parallel algorithm development scheme

[1] In spite of the fact that for many scientific and technical problems not only sequential but also parallel solving methods are known, this assumption is, of course, strong. Actually the algorithm development process for the newly emerging problems, which require time-consuming computations, is a considerable part of all the work performed.

After carrying out all the design stages mentioned above, it is possible to evaluate the efficiency of the developed parallel methods. For this purpose the quality characteristics for the generated parallel computations should be evaluated (speedup, efficiency, scalability). It may appear to be necessary to repeat some (in the limiting case even all) design stages according to the results of the analysis. It should be mentioned that the return to the previous design stages may happen at any stage of parallel computational scheme design.

In this respect the additional action, which is repeated frequently in the design scheme described above, is the adjustment of the number of the formulated subtasks after the available number of processors has been defined. The subtasks may be aggregated, if only a small number of processors are available, or vice versa subdivided. In general these actions may be considered as *scaling* the developed algorithm and may be added as a separate stage of parallel computation design.

To apply the parallel method, which is eventually obtained, it is necessary to develop programs for solving the formulated set of subtasks and distribute these programs among the processors in accordance with the selected distribution scheme. The developed program code must ensure solution of the subtask set. To put this into practice, one will, for example, have to implement solution for each subtask of a program, but more often one writes a program encorporating all steps required to solve all the subtasks. Such a program code (*metaprogram*) is developed so that the program, depending on control parameters, is able to be set for solving the required subtask (the computing element number can be used as the control parameter). For the purpose of computing, this metaprogram can be copied for all computing elements – such approach is used by MPI for distributed memory multiprocessor systems; programs executed for different computing elements are usually called *processes*. Metaprograms can be used to generate sets of separate command *threads* – this happens in case when OpenMP runs on shared memory systems.

The parallel program is run for the purpose of computation. For information exchange, the parts of program (processes or threads) executed in parallel must have access to data transmission facilities (*message channels* in case of distributed memory systems or *shared variables* for shared memory systems).

Each computing element (CPU or CPU core) of the system is usually dedicated to solve one and only subtask; however, in case of numerous subtasks or restricted number of computing elements this rule cannot be observed, which results in simultaneous execution of several parallel program sections (processes or threads) by the computing elements. Specifically, for development and initial check of a parallel program, one computing element can be used to execute all

its parallel parts (if placed within the same computing elements, the parallel program parts are time-shared).

It should be noted that the developed design and implementation system for parallel computations was initially intended for distributed memory systems where communication is based on messaging by processors via communication channels. However, this scheme may also be applied to shared memory systems without losing parallel computation efficiency: in this case, mechanisms of passing messages are replaced by operations of accessing shared variables. To make further educational materials less complex, *the scheme of design and implementation of parallel computations will be described as applicable to shared memory systems*.

This Section has been written based essentially on the teaching materials given in Foster (1995) and Quinn (2004).

This Section has been written based essentially on the teaching materials given in Kumar, et al. (1994), Pfister (1995) and Quinn (2004).

The lecture is dedicated to the described method for parallel algorithm development.

Test questions

1. What are the basic stages of the methodology of parallel computation design and development?
2. How is the "subtasks-messages" model defined?
3. How is the "processors-channels" model defined?
4. What basic requirements should be met in parallel algorithm development?
5. What are the basic operations at the stage of subtask selection?
6. What are the basic operations at the stage of analyzing information dependencies?
7. What are the main operations at the stage of scaling the available subtask set?
8. What are the main operations at the stage of distributing subtasks among the processors of a computer system?
9. How does the "manager-worker" scheme provide dynamic management ofe computational load?
10. Which parallel computation method was developed for solving the gravitational problem of *N* bodies?

Practice

1. Design a scheme of parallel computations using the methodology described in the lecture for designing and developing parallel methods:

- For the problem of searching the maximum value among minimal elements of matrix rows (such calculations occur in solving the problems of matrix games):

$$y = \max_{1 \le i \le N} \min_{1 \le j \le N} a_{ij},$$

(pay special attention to the situation when the number of processors exceeds the matrix order, i.e. $p>N$),

- For the problem of computing a definite integral using the method of rectangles:

$$y = \int_a^b f(x)dx \approx h\sum_{i=0}^{N-1} f_i,\ f_i = f(x_i),\ x_i = i\,h,\ h = (b-a)/N\ .$$

Reference

1. **Andrews, G. R.** (2000). Foundations of Multithreaded, Parallel, and Distributed Programming.. – Reading, MA: Addison-Wesley
2. **Bertsekas**, **D.P., Tsitsiklis, J.N.** (1989) Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.
3. **Buyya, R.** (Ed.) (1999). High Performance Cluster Computing. Volume1: Architectures and Systems. Volume 2: Programming and Applications. - Prentice Hall PTR, Prentice-Hall Inc.
4. **Kahaner, D., Moler, C., Nash, S.** (1988). Numerical Methods and Software. – Prentice Hall
5. **Foster, I.** (1995). Designing and Building Parallel Programs: Concepts and Tools for Software Engineering. Reading, MA: Addison-Wesley.
6. **Quinn, M. J.** (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.
7. **Wilkinson, B., Allen, M.** (1999). Parallel programming. – Prenrice Hall.S