Part 3.

Methods of parallel computations

6. 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.

[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.

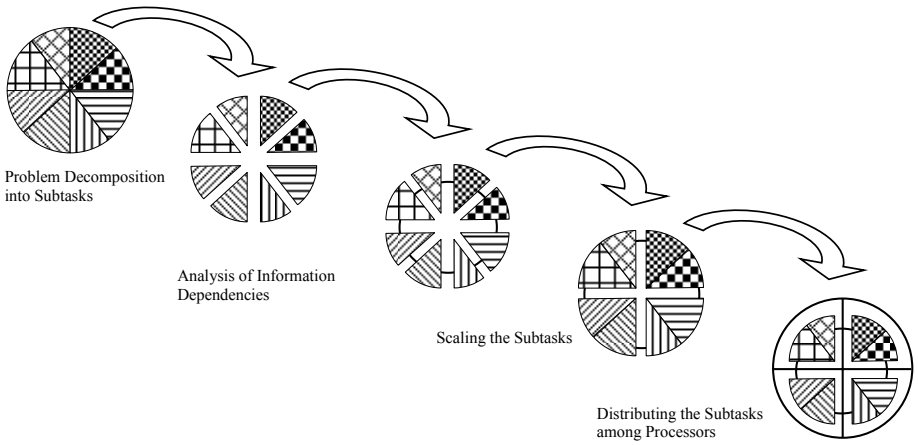

Figure 6.1. Parallel algorithm development scheme

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 the developed programs among the processors in accordance with the selected distribution scheme. The programs are started to do the computations (at the execution stage the programs are called *processes*). To provide the required interactions, the program should have the routines for information passing between subtask and the computer system should have some communication channels between processors.

It should be mentioned that each processor is usually appointed for solving a single subtask. However, if there are many subtasks or the number of processors in use is limited, this rule may not be observed. As a result, sevenral programs (processes) may be executed on each processor. In particular, when a parallel program is being developed or is being tested (at the initial development stage), one processor may be used for execution of all processes (in this case processes are executed in the time-sharing mode).

If we consider the developed scheme of parallel computation design and implementation carefully, we will see that this approach is to a considerable extent designated for the distributed memory computer system. The information interactions are realized in such systems by means of message passing along the communication channels among the processors. Nevertheless, this scheme may also be used without any loss of parallel computation efficiency for developing the parallel methods for shared memory systems. In this case the mechanism of message passing for providing the information interactions should be changed by operations of access to the shared variables.

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).

6.1. Parallel Program Modeling

The described scheme of parallel computation design and implementation helps to understand parallel programs and algorithms. At the design stage the parallel method may be represented as *the "subtasks - messages"* graph. This graph is the aggregated representation of the information dependencies graph (the "operations – operands" graph – see section 2). Analogously to it we may use the model in the form of *the "processes – channels" graph* to describe the parallel program at the execution stage. In this model we use the concept of processes instead of subtasks, and the information dependencies are substituted for message communication channels. In addition, this model helps to demonstrate the distribution of processes among the

computer system processors in case when the number of subtasks exceeds the number of processors (see Figure 6.2)

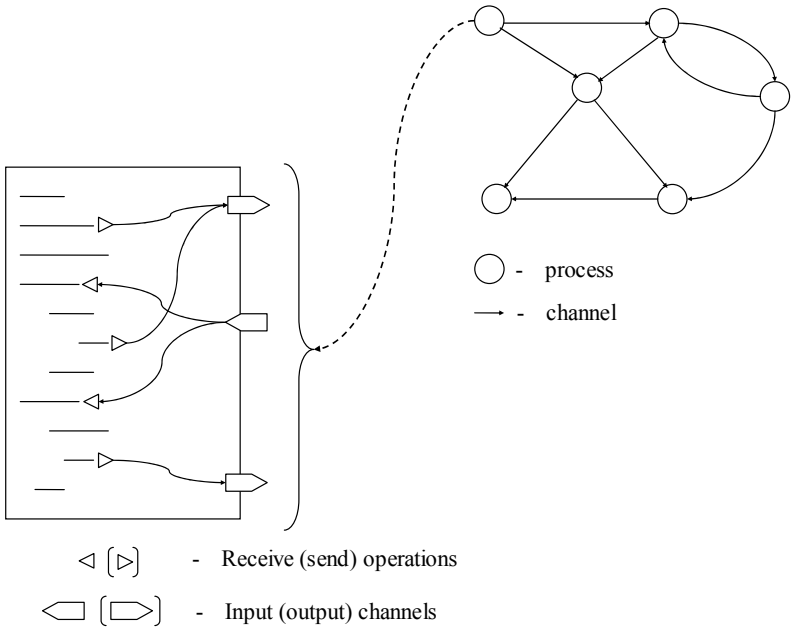

Figure 6.2. Parallel program model in the form of the "processes-channels" graph

The use of the two parallel computation models [2)] helps to classify the problems, which appear in parallel method development. The first model – the "subtasks - messages" graph - allows us to concentrate on the problems of formulating the subtasks of equal computation complexity, providing also a low level of information dependences among the subtasks. The second model – the "processes–channels" graph - concentrates on the problems of distributing subtasks among the processors. It provides an additional possibility to decrease the complexity of information interactions among the subtasks by assigning a group of high frequently interacting processes to the same processor. This model besides helps to analyze the efficiency of the developed parallel method better and provides the possibility to describe the process of parallel computation execution more adequately.

Now we will give some additional explanations for the concepts used in the processes-channels model:

- *A process* is any program, which is executed on a processor. The process uses the processor's local memory and contains a number of data passing operations for interacting with other parallel executed processes,
- *A data transmission channel* may be considered from the logical point of view as a message queue, to which one or more processes may send the transmitted data, and from which the addressee process can receive the messages sent by the other processes.

It may be generally considered that channels appear dynamically at the moment when the first communication operation of the channel is executed. It can be assumed also that a channel may correspond to one or several commands of data reception of the addressee process. Analogously a channel may be used by one or several data transmission commands of one or several processes. To decrease the complexity of parallel method modeling and analysis, it will be proposed that channel capacity is unlimited. As a result, data communication operations are carried out practically without any delays by means of simple copying messages into a channel. On the other hand, message reception operations may lead to delays (blocking), if the data, requested from the channel, has not been sent yet by the processes.

The "processes-channels" model has a very important advantage. There is a distinct separation of local (carried on a separate processor) computations and interacting the simultaneously executed processes. This approach significantly decreases the complexity of efficiency analysis for parallel methods and considerably simplifies the problems of parallel program development.

[2)] In Foster (1995) only one model is considered. This "task – channel" model can be considered as some intermediate technique in comparison to the models described in this Section. For instance, the possibility of using one processor for solving several subtasks simultaneously is not taken into account in the "task - channel" model.

6.2. Stages of Parallel Algorithm Development

Let us consider the methodology described above in more detail. This methodology is based on the approach firstly developed in Foster (1995). As it has already mentioned parallel to develop a parallel method we have to determine subtasks, find out the information dependencies, scale and distribute subtasks among the computer system processors (see Figure 6.1). To demonstrate the formulated methodology, we will further analyze the problem of searching the maximum value among matrix elements (for instance, such calculations are executed for solving linear equation systems by means of the Gauss method):

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

This problem is not very difficult and is formulated for purely illustrative purposes. In addition we will consider the development stages and describe the use of the methodology for parallel algorithm development in more complicated case. Besides this development scheme will further be applied for the description of all other parallel computation methods.

6.2.1. Computation Decomposition into Independent Subtasks

To decide what subtasks have to be formulated the computation scheme of solving the studied problem should be analyzed. The selected approach has to meet are the following requirements: it should be able to provide the equal amount of computations in the selected subtasks and the minimum information dependencies among these subtasks (under the same conditions a small number of large data communications should be preferred to the frequently executed short communications). Generally, the analysis and the subtask decomposition is a rather complicated problem. It may be solved more easily in case of the two frequently occurring types of computational schemes:

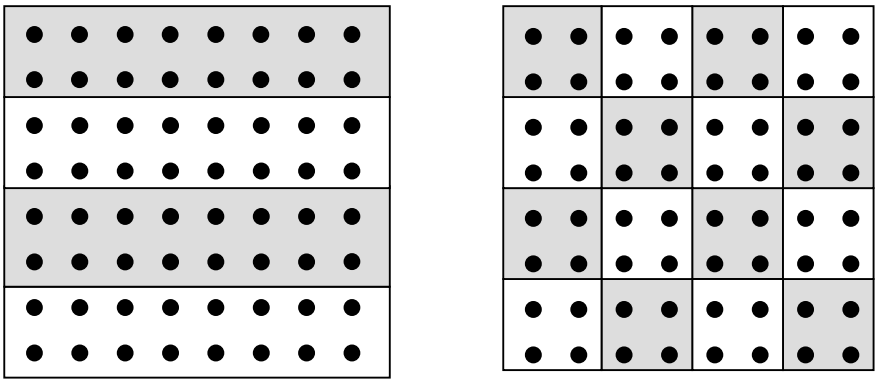

Figure 6.3. Matrix data decomposition: a) a striped scheme, b) a checkerboard block scheme

- For a wide class of problems the computations come up to carrying out the uniform processing of a large set of data. Matrix computations, numerical methods of solving the partial derivative equations and so on may be referred to this class. This is the case of the so-called *data parallelism*, and subtasks selection comes up to partition of the available data. Thus, for instance, for our problem of searching the maximum value the matrix *A* may be partitioned into separate rows (or the groups of consecutive rows) – a *striped scheme* of data decomposition (see Figure 6.3) - or it may be separated into rectangular sets of elements – a *checkerboard block scheme* of data decomposition. For a large number of problems, data decomposition leads to creating one -, two-, three-dimensional sets (*grids* or *meshes*)of subtasks. The information links in these subtasks exist only among the nearest neighbors.

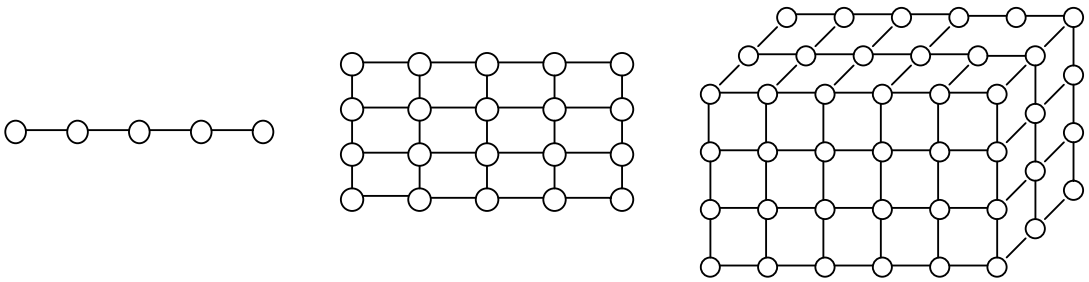

Figure 6.4. Data decomposition into regular one-, two-, and three-dimensional sets of basic subtasks

- For the other part of the problems the computations can consist in carrying out different operations over the same set of data. This is the so-called *functional parallelism* (the examples may be the problems of processing the sequence of queries to the information data bases, the computations with the simultaneous use of different computational algorithms etc.). Functional decomposition is frequently used for organizing the pipeline data processing (thus, for instance, in carrying out some data transformation the computations may be reduced to functional sequence of data input, data processing and data output).

The choice of the adequate *decomposition level* is an important issue in formulating subtasks. Generating the maximum possible amount of subtasks provides the use the maximum achievable parallelism level for the problem being solved. However, it complicates the parallel computation analysis. The use of only large-scale subtasks in computation decomposition leads to a clear parallel computation scheme. However, it can make it more difficult to use a large number of processors efficiently. A reasonable combination of these two approaches may consist in the use of only those subtasks, for which the methods of parallel computations are known as *constructive decomposition elements*. Thus, for instance, analyzing the problem of matrix multiplications, it is possible to formulate subtasks as the calculations of inner product or matrix-vector multiplication. This intermediate method of decomposing computations makes it possible to provide both the simplicity of computation decomposition scheme and the efficiency of parallel calculations. The computations selected with the help of this approach will be further referred to as *the basic computational subtasks*. They may be *elementary* (nondivisible), if they do not allow further partitioning, or *aggregated* if otherwise.

For the problem discussed here sufficient decomposition level may consist, for instance, in subdividing matrix A into a set of separate rows and obtaining on this basis a set of subtasks for maximum value search in separate rows. The structure of the information links generated in this case corresponds to the linear graph - see Figure 6.5

The list of questions given in Foster (1995) may be suggested for estimating the correctness of the computation decomposition stage:

- Does the performed decomposition increase the amount of computations and the necessary memory size?
- Is the balanced loading of all the available processors possible with the chosen decomposition approach?
- Are the selected subtasks sufficient for efficient loading of the available processors (taking into account the possibility to increase their number)?

6.2.2. Analysis of Information Dependencies

Finding out information dependencies among subtasks does not usually cause many problems. However, it should be noted that it is actually difficult to separate the stages of subtask selection and information analisys. Subtask selection must be carried out with regard for the emerging information links. It may appear to be necessary to repeat the computation decompostion stage after completing the analysis of the volume and frequency of the necessary data communication among subtasks.

The following forms of information interaction should be differentiated in analyzing the information dependencies (the preferable information interaction forms are underlined):

- *Local and Global schemes* of data communication – for local schemes the data communication s at every moment are executed only among a small number of subtasks (placed, as a rule, on neighboring processors). For global data communication operations all subtasks participate in the communication process;
- *Structural and Arbitrary interaction methods* – for structural methods interactions leads to forming some standard communication schemes (for instance, in the form of a ring, a rectangular mesh etc.). The scheme of data communication operation for arbitrary interaction structures is not homogeneous;

- *Static or Dynamic schemes* of data communication – for the static schemes the information interaction moments and participants are fixed at the stages of design and parallel program development. For the dynamic variant of interaction the structure of data communication operation is determined in the course of computations;
- *Synchronous and Asynchronous interaction methods* – data communication operations for synchronous methods are carried out only if all the interaction participants are ready for interaction and end only after the completion of all the communication actions. If the operations are carried out asynchronously, the interaction participants do not have to wait for the total completion of data communication actions. It is rather difficult to choose the preferable form of data communication for these interaction methods: the synchronous mode is, as a rule, simpler to use, while the asynchronous one makes possible to considerably decrease the time delays caused by information interaction operations.

As it has been already mentioned in the previous section, for the problem of maximum value search the structure of information links looks as shown in Figure 6.5, if the subtasks have been selected as calculations of maximum value in separate rows of the matrix *A*.

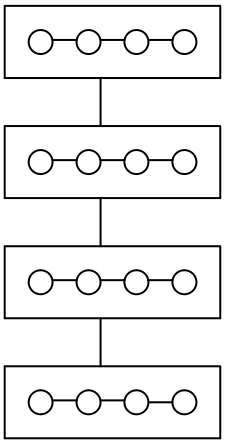

Figure 6.5. The structure of the information links for the problem of maximum value search

As in the previous case, estimating the stage of analyzing the information dependencies it will be useful to examine the questions suggested Foster (1995):

- Does the computational complexity of subtasks correspond to the intensity of their information interactions?
- Is the intensity of information interactions the same for different subtasks?
- Is the information interaction scheme local?
- Will the formed information dependencies hinder parallel execution of subtask computations?

6.2.3. Subtask Set Scaling

Developed computing scheme of parallel computations has to be scaled, if the number of available subtasks differs from the number of processors planned to be used. To decrease the number of subtasks, it is necessary to perform the aggregation of calculations. The rules applied in this case coincide with the recommendations of the computation decomposition stage. The anew defined subtasks should have close computational complexity and the volume and the intensity of information interactions among subtasks should stay at the minimum possible level. As a result the first candidates for aggregation are subtasks with a high degree of information interdependence.

If the available set of subtasks is not enough to load all the available processors, it is necessary to perform decomposition of computations. As a rule, such decomposition does not cause any difficulties, if the methods of parallel computations for basic subtasks are known.

In general the scalability stage should produce rules for subtask aggregation and decomposition, which should parametrically depend on the number of the processors used for computations.

For the problem of maximum value search discussed here the aggregation of computations may consist in combining of separate rows into groups (a striped scheme of matrix separation – see Figure 6.3a). The rows of the original matrix *A* may be subdivided into several parts (blocks) if subtask decomposition is required.

The list of questions suggested in Foster (1995) for evaluation of the scalability stage correctness looks as follows:

- Will the computation locality deteriorate after scaling the available subtask set?

- Are the subtasks after scalability of the same computation and communication complexity?
- Does the number of subtasks correspond to the number of the available processors?
- Do scalability rules depend parametrically on the number of processors?

6.2.4. Subtask Distribution among Processors

Distributing the subtasks among processors is the final stage of parallel method development. It should be noted that the management of load distribution for processors is possible only for computer system with the distributed memory. For multiprocessors (the systems with shared memory) process distribution is usually performed by the operating system automatically. Besides this stage of distributing subtasks among processors is redundant, if the number of subtasks coincides with the number of available processors, and the data communication network topology of a computer system is a complete graph (i.e. all the processors are connected by direct communication links).

The basic criterion of the success for this stage is the efficiency of processor utilization, which is defined as the relative fraction of time, during which the processors were being used for computations connected with solving the original problem. The ways to achieve good results in this direction remain the same. As previously, it is necessary to provide uniform distribution of computational load among processors and minimize the number of messages transmitted among processors. Exactly as it was at earlier design stages, the optimum solution of subtask distribution problem is based on the analysis of "subtasks-messages" graph connectivity. Thus, for instance, the subtasks, among which there are information interactions, should be located on the processors connected by direct data communication lines.

It should be noted, that the requirement to minimize the information communications among processors may contradict to the condition of uniform processor loading. For instance, we may place all subtasks on a processor and eliminate interprocessor message communications completely. It is obvious however, that for the majority of processors the loading will be minimal in this case.

For the problem of maximum value search, distributing subtasks among processors does not cause any difficulties. It is enough to provide placement of the subtasks, among which there are information links, on the processors for which there are direct data communication channels. As the structure of informational links for the problem discussed here looks as a linear graph, this requirement can be met in case of practically any network topology of the computer system.

The problems of balancing the computational load become more complicated, if the computation scheme can be changed in the course of solving the problem. The reasons for this may be, for instance, nonuniform grids when partial derivative equations are solved, sparse matrice computations, etc.[3)]. Besides, the subtask complexity estimations used at design stages may be evaluated approximately, and, finally, the number of subtasks may be vary in the course of computations. In these cases it may become necessary to redistribute the basic subtasks among processors immediately in parallel program execution (or, as they say, to carry out *dynamic balancing* of computational load). These issues are among the most complex (and interesting) ones in the field of parallel computations. Unfortunately, discussion of these issues lies beyond the scope of the Section (additional information may be found, for instance, in Buyya (1999) and Wilkinson and Allen (1999)).

As an example we will briefly describe the widely used method of load dynamic management, which is usually termed as *manager-worker scheme.* The use of the approach supposes that subtasks can be generated and terminated in the course of computations. It is also assumed that there are either no information interactions among the subtasks or their amount is minimal. In accordance with the manager-worker scheme a single processor-manager is selected in the system. This processor-manager has access to the information of all the available subtasks. The rest of the processors are workerrs. They apply to processor-manager to obtain the computation load. The new subtasks generated in the process of the computations are transmitted back to the processor-manager and may be obtained for solving, if processors-workers apply to the processor-manager. Completion of computations occurs as soon as the processors-workers have ended the solving process of all the subtasks transmitted to them, and the processor-manager does not have any computation jobs for execution.

The list of questions suggested in Foster (1995) for checking subtasks distribution stage is as follows:

- Will the placement of several tasks on one processor lead to the increase of additional computational overhead?
- Is it necessary to balance the calculations dynamically?
- Cannot the processor-manager be a bottleneck in manager-worker scheme application?

[3)] It can be noted that even for our simple problem we can observe different computation complexity of the formulated basic subtasks. Thus, for instance, the number of operations for search of the maximum value of a row where the first element has the maximum value and the row in which the values are placed in ascending order will differ twice.

6.3. Parallel Solving the Gravitational Problem of N Bodies

Many computational problems in the field of physics can be reduced to data processing operations for each pair of objects of the available physical system. This group of problems is represented by the well-known *gravitational problem of N bodies* (or simply *the problem of N bodies*) - see, for instance, Andrews (2000). In the most general way the problem may be described as follow.

Let there be a great number of bodies (planets, stars, etc.). The mass, the initial position and the velocity are known for each of the bodies. The position of the bodies is changed continually due to the influence of gravitation. The problem is to compute all the system's changes during a certain given time interval. To provide such computations the time interval is usually divided into short time intevals. Further for each time interval the forces affecting each body are computed. Then the velocities and positions of the bodies get renewed.

The evident algorithm for solving the problem of N bodies consists in carrying out all the necessary computations for each pairs of bodies. As a result, the time of one iteration execution will be the following[4)]:

$$T_1 = \tau N(N-1)/2,$$

where τ is the time of parameter recalculation for a pair of bodies.

The description shows that the computation scheme of the described algorithm is simple enough. It makes possible to use the problem of N bodies as an illustration of applying the approaches to parallel algorithm development.

6.3.1. Computation Decomposition into Independent Subtasks

The choice of computation decomposion method does not cause any difficulties. The evident approach is to choose all the set of computations connected with processing data of a physical system's body as the basic computational subtask.

6.3.2. Analysis of Information Dependencies

Calculation execution connected with each subtask becomes possible only if data (position and velocity) concerning all the bodies are available in the subtasks. As a result, before each calculation iteration start, each subtask must get all the necessary data from the other subtasks of the system. This procedure of data communications, as it was pointed out in section 3, is called *single-node gather*. In the algorithm discussed here this operation must be executed for each subtask. This variant of data communications is usually called *multi-node gather* or *all gather*.

Defining requirements to the necessary results of information communications does not uniquely lead to determing the necessary algorithms of information communications among subtasks. Obtaining the required results may be provided through various algorithms of multi-mode gather operation execution.

The simplest way to carry out the necessary information communications is to execute a sequence of steps, at each of which all the available subtasks are splitted into pairs, and the data communications are carried out between the subtasks, which form these pairs. If the pairwise subtask division is proper, then (*N-1*) iterations of the described above procedure will lead to complete implementation of the required data gather operation.

The described above method of information communications is rather time-consuming. To gather all the necessary data, (*N-1*) iterations are required. (*N/2*) data communication operations are performed simultaneously at each iteration stage. To decrease the required number of iterations, it is worthwhile to pay attention to the fact that after completing the iteration 1 of data gather operation, the subtasks will contain not only their own data, but also the data of their matches (the subtasks they formed pairs with). As a result, at the second iteration of data gathering it will be possible to form pairs of subtasks to exchange data simultaneously about two bodies of the physical system. Thus after completing the second iteration each subtask will contain the information about four bodies of the system, etc. It is evident that this method of information communications allows executing all the necessary communications in log_2N iterations. It should be noted that the amount of data transmitted in each exchange operation is doubled from iteration to iteration. At the first iteration the data about only one body is transmitted between the subtasks, at the second iteration – about two bodies, etc.

The application of this method of multi-node gather operation implementation leads to the definition of information link structure among the subtasks as a N-dimensional hypercube.

[4)] It should be noted that there are more efficient sequential algorithms for solving the problem of N bodies. However, studying them may require considerable efforts. With regard to these circumstances this particular "evident" but not the fastest method is chosen for further consideration though generally the best computation schemes should be chosen for parallelizing.

6.3.3. Scaling and Distributing Subtasks among Processors

As a rule, the number of bodies in a physical system N exceeds the number of processors p. As a result, the subtasks discussed previously must be aggregated. They should be united in the framework of one computation subtask for the group of (N/p) bodies. After this aggregation the number of subtasks and the number of processors will coincide. Distributing these subtasks among processors is evident, it will only be necessary to use processors with direct communication lines for subtasks, which execute information communication of data gather operations.

6.3.4. Efficiency Analysis of Parallel Computations

Let us estimate the efficiency of the developed parallel computation methods for solving the problem of N bodies. As the suggested variants differ from each other only by the methods of carrying out information communications, it is enough to define the duration of multi-node gather operation in order to compare them. Let us use the model suggested by Hockney (see section 3) to evaluate the message communication time. The duration of data gather operation for the first parallel computation method may be expressed as:

$$T_p^1(comm) = (p-1)(\alpha + m(N/p)/\beta),$$

where α, β are the parameters of the Hockney model (latency and bandwidth of data communication network), and m sets the transmitted data size for one body of the physical system.

For the second information communication method, as it has already been mentioned previously, the sizes of the transmitted data at different iterations of data gather operation are different. At the first iteration the size of the transmitted data is (mN/p). At the second iteration this size doubles and appears to be equal to 2(mN/p), etc. Generally, for iteration number i the data size is estimated as 2^{i-1}(mN/p). As a result, the duration of data gather operation execution in this case may be defined by the following equation:

$$T_p^2(comm) = \sum_{i=1}^{\log p}(\alpha + 2^{i-1} m(N/p)/\beta) = \alpha \log p + m\,(N/p)(p-1)/\beta\,.$$

The comparison of the obtained expressions shows that the second developed parallel computation method is far more efficient, involves fewer communication overhead and allows better scalability, if the number of available processors is increased.

6.4. Summary

In this chapter we discussed the methodology of parallel algorithm development suggested in Foster (1995). This methodology includes the stages of subtask selection, determination of information dependencies, the stages of scaling and distributing subtasks among the processors of a computer system. Applying this methodology we assume that the computation scheme of solving the discussed problem is already known. The main requirements, which must be met in developing parallel algorithm, are to provide with uniform processor load and low information interactions among the formulated subtasks.

We discussed two models to describe the computation parallel scheme obtained in the course of development. The first of them is “subtasks-messages” model, which can be used at the stage of parallel algorithm design. The second one is “processes-channels” model, which may be applied at the stage of parallel program development.

In conclusion we demonstrated the application of the considered methodology to develop parallel methods for solving the gravitational problem of N bodies.

6.5. References

The methods discussed in this chapter were first proposed in Foster (1995). This work considers the methodology in more detail. Besides, it contains several examples of their application for parallel method development for solving a number of computation problems.

The information given in the work by Quinn (2004) may appear to be useful in consideration of the issues connected with design and development of parallel algorithms.

The gravitational problem of N bodies is considered in detail in Andrews (2000).

6.6. Discussions

1. What are the initial assumptions for the methodology of parallel algorithm development discussed in the Section?

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

6.7. Exercises

1. Perform the implementation of a cascade scheme of computing a sum of numerical values (see the Section 2) and compare the execution time of this program and and the function *MPI_Bcast* of MPI.

2. Implement the discussed methods of multi-node gather operation and compare their execution time. Associate the obtained time characteristics with the available theoretical estimations. Compare them with the time of executing the function *MPI_Allgather* of MPI.

3. Design a scheme of parallel computations using the methodology described in the Section 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:

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

(the description of this numerical integration method is given, for instance, in Kahaner, Moler and Nash (1988))

4. Develop parallel programs for the proposed parallel methods in the Exercise 3.

5. Design the scheme of parallel computations for the problem of matrix-vector multiplication using the methodology of designing and developing parallel methods, described in Section.

References

Andrews, G. R. (2000). Foundations of Multithreaded, Parallel, and Distributed Programming.. – Reading, MA: Addison-Wesley

Bertsekas, **D.P., Tsitsiklis, J.N.** (1989) Parallel and distributed Computation. Numerical Methods. - Prentice Hall, Englewood Cliffs, New Jersey.

Buyya, R. (Ed.) (1999). High Performance Cluster Computing. Volume1: Architectures and Systems. Volume 2: Programming and Applications. - Prentice Hall PTR, Prentice-Hall Inc.

Kahaner, D., Moler, C., Nash, S. (1988). Numerical Methods and Software. – Prentice Hall

Foster, I. (1995). Designing and Building Parallel Programs: Concepts and Tools for Software Engineering. Reading, MA: Addison-Wesley.

Quinn, M. J. (2004). Parallel Programming in C with MPI and OpenMP. – New York, NY: McGraw-Hill.

Wilkinson, B., Allen, M. (1999). Parallel programming. – Prenrice Hall.