The ParaLab System for Investigating the Parallel Algorithms

Victor Gergel and Anna Labutina

Nizhni Novgorod State University
gergel@unn.ru, anna.labutina@cs.vmk.unn.ru

Abstract. In this paper we introduce a software system which allows to carry out and visualize computational experiments for studying and researching the parallel algorithms of solving complicated computational problems in imitation mode on one single sequential computer. User can "assemble" a parallel computational system of cluster type that consists of multiprocessor and multicore nodes connected with the network, set up the problem to be solved, carry out the parallel solving algorithm, collect and analyze the results of computational experiments. To estimate the execution time of parallel method on current hardware system we use the sophisticated models. For every implemented parallel method we proved the theoretical estimations of the execution time by comparing the real time of the execution on the NNSU high performance cluster with the time, that can be calculated using the model.

Keywords: high performance computing, parallel computing, parallel computations modeling, cluster, multiprocessor architecture, multicore architecture.

1 Introduction

The development of the computer architecture and network technologies, together with investigations of new time-consuming scientific and applied problems that demand massive computations showed high necessity of parallel computations, made high performance computing the cornerstone of programming and computational technology.

But despite of science needs and the actuality of parallel computations, so far they are not as widely used as it was predicted. One of the possible reasons is the necessity of developing new parallel algorithms to solve the new computationally intensive problems. It is well-known that the speedup of solving the task on parallel computational system can only be achieved when the algorithm is divided into set of independent processes that can be run simultaneously. The other reason is that the debugging of parallel code is a high complexity problem, which makes it necessary to fully understand the behavior of the system of computational processes run in parallel. That is why competence in modern high performance computational system design trends, in new tools developed to achieve parallelism, the ability to create models and methods for solving the problems in parallel are the major qualities for specialists in applied mathematics, computer science and information technologies.

C.H. Hsu and V. Malyshkin (Eds.): MTPP 2010, LNCS 6083, pp. 95–104, 2010.

2 Working with ParaLab

While working with ParaLab the user has an access to a wide range of tools to set the computational experiment parameters. She can model the computational system, chose the problem, carry out the parallel algorithm, collect and analyze the results of computational experiments.

2.1 Modeling the Parallel Computational System

ParaLab allows to simulate the parallel computational experiments execution on multiprocessor (*SMP*) and *multicore* architectures. The computational system appears to consist of the *computational nodes* (*computers*). Each node has one or more *processors*, and each processor has one or more *cores*. The ParaLab system architecture doesn't limit the maximum amount of cores in processor and processors in one node, but for the sake of visualization we limit the number of cores to be equal to 1, 2 or 4 and number of processors to be 1 or 2.

In order to simulate the computer system, it is necessary to determine the network topology, the number of computational nodes, the number of processors and cores in one node, the performance of each core, and the characteristics of the communication network (the latency, the bandwidth and the data communication method). It should be noted that the computer system is assumed to be homogeneous in the ParaLab system, i.e. all the computational nodes have the equal amount of processors, every processor consists of the same number of cores, cores possess equal performance, and all the communication lines have the same characteristics.

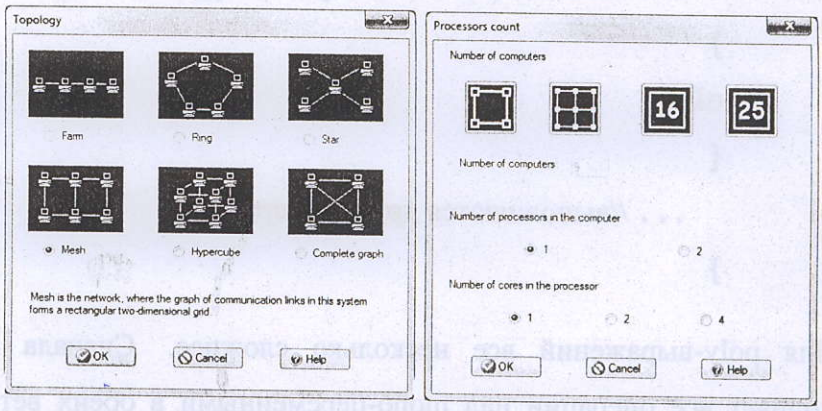

Fig. 1. Dialog windows to set up the computational system parameters

The data communication network *topology* is defined by the structure of communication lines among the computer system processors. The system ParaLab supports the following network topologies: farm, ring, star, grid, hypercube, full graph (clique).

The system ParaLab allows user to set the desirable number of nodes for the selected topology. The choice of the system configuration is performed in accordance with the type of the topology used.

The *performance* of the core in the ParaLab system is measured by the number of floating point operations per second (flops). It should be noted that to estimate the execution time of the experiment, it is assumed that all the computer instructions correspond to the same floating point operation.

The time of data transmission among the processors determines the *communication overhead* of the parallel algorithm execution in a multiprocessor system. The main set of parameters, which makes possible to estimate the data communication time, contains the following values:

– *latency* (α). It is the time, which characterizes the duration of preparing a message for transmission;

– *network bandwidth* (β). It is defined as the maximum amount of data, which can be transmitted in a certain unit of time through a data communication channel.

Among the data communication methods, implemented in ParaLab, there are the following two well-known communication methods [3]. The first method is aimed at *passing messages* as indivisible information blocks (*store-and-forward routing* or *SFR*). The second communication method (*cut-through routing* or *CTR*) is based on representing the transmitted messages as a set of information blocks of smaller sizes (*packets*).

2.2 Selecting the Problem and the Parallel Method

The following widely used parallel algorithms applied to solving complicated computational problems in various scientific and technical applications are implemented in the system ParaLab: the algorithms for data sorting, the algorithms for matrix operations, the algorithms for solving the systems of linear equations, graph processing, the algorithms for solving differential equations in partial derivatives and the algorithms for global multiextremal optimization.

As a rule, for every task there are several parallel solving methods implemented. For the matrix-vector multiplication task we implemented algorithms based on block, rowwise and columnwise matrix decomposition. For the matrix multiplication problem there are parallel Fox's and Cannon's algorithms and the algorithm based on striped matrix decomposition. For the problem of solving the system of linear equations we present the parallel variants of Gauss method and conjugate gradient method. For the sorting problem we implemented parallel variants of bubble sort, Shell sort and quick sort. For the graph processing task there are parallel algorithm for building minimal spanning tree, Dijkstra's and Floyd's algorithms for shortest paths problem. For the problem of solving the differential equation in partial derivatives we have parallel Gauss-Seidel algorithm. Parallel index method is implemented for the problem of multiextremal optimization.

The main problem parameter in ParaLab is the amount of the initial data. User can set additional parameters for some types of problems. For example there is a possibility to choose the boundary conditions for the problem of solving the differential equation in partial derivatives, to choose the type of function for the

problem of multiextremal optimization, to create a graph with the help of built-in graph editor for the graph processing problem.

2.3 Carrying Out the Computational Experiment

ParaLab provides various forms of graphical demonstration of parallel computation results in order to observe the process of carrying out a computational experiment of solving complicated time consuming computational problems. Before the parallel algorithm execution user can set the visualization parameters for demonstration speed, the mode of communication operation visualization, the required level of details to be shown.

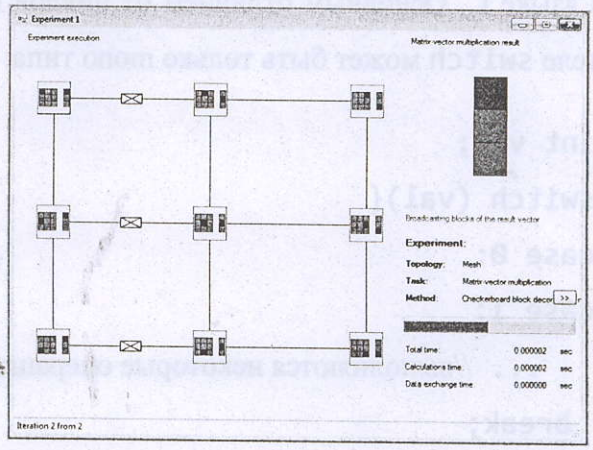

Fig. 2. The window of the computational experiment while solving the problem of matrix-vector multiplication

The system ParaLab provides different schemes of carrying out experiments to give convenient possibilities for studying parallel algorithms of solving complicated computational problems. Problems may be solved in the sequential execution mode, in the time sharing mode with the possibility to simultaneously observe the algorithm iterations in all the computational experiment windows. Carrying out series experiments, which requires long-continued computations, may take place in the automatic mode. Experiments may be also carried out in the step-by-step mode.

2.4 Accumulating and Analyzing the Experiment Results

To accumulate the results of the executed experiments, ParaLab provides a special memory, which is hereinafter referred to as the *experiment log*. The results are stored in the experiment log automatically. Accumulated results can be used for observing and analyzing.

For the experiments saved in the experiment log, we build the graph that shows how the execution time and the speedup depend on problem and computational system parameters. These graphs are built in accordance with the theoretical models we use to estimate the execution time of the parallel algorithm.

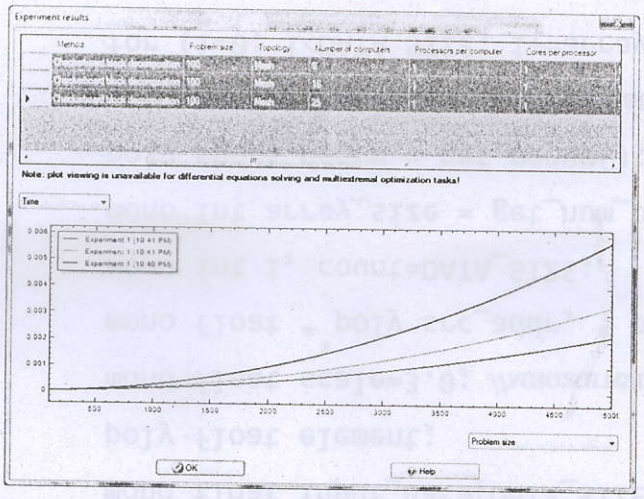

Fig. 3. The experiment log window

3 Modeling Parallel Computations

3.1 Model for the Local Computations

While creating a model to estimate the time of local computations we assume that this time is the sum of the calculation time and the memory access time:

$$T_1 = T_{calc} + T_{mem} \quad (1)$$

Here the calculation time is the result of multiplication of the executed operations number N by the time of one operation execution τ. The memory access time is the result of division of the maximum amount of data M by the memory bandwidth β. To make the estimation more precise we should consider that the data comes from memory not in byte-by-byte mode but in full cache lines, the length of one cache line is equal to L bytes. The worst case is when every data element should be downloaded from the memory and it falls in the separate cache line. We should also consider the RAM latency α that can significantly influence the time of computations. Thus, the model for the local computations execution time can be the following:

$$T_1 = N \cdot \tau + M \cdot (\alpha + L/\beta) \quad (2)$$

This model doesn't reflect the modern processor architecture, where the processor has small but fast local memory, which is called cache memory. In order to get the fast access to the necessary data this data is downloaded from RAM to cache before the computations with the use of different prediction algorithms. This download can be performed simultaneously with computations and doesn't affect the time of computation execution. The situation when the necessary data is not in the cache and the processor should wait for them to be downloaded from RAM is called *cache miss*. To make the model of computational time more precise we need to know the number

of cache misses appeared during computations. With this new information we can correct the time that the processor spends on waiting for the data to be downloaded from the RAM:

$$T_1 = N \cdot \tau + \gamma M(L/\beta + \alpha) \quad (3)$$

where γ is the cache miss ratio (number of cache misses divided by the number of cache access operations), which can be theoretically estimated.

To make a decision about the model accuracy the computational experiments were carried out on the computer with the Intel core 2 quad Q6600 processor. The architecture of this processor includes first-level caches with the bandwidth of 153 Gb/sec and latency of 1,22 nsec. The RAM of the target system has a bandwidth of 12,4 Gb/sec and latency of 8,31-80 nsec. The algorithm of matrix-vector multiplication was executed. The code for this algorithm is the following:

```
for (i=0; i<Size; i++) {
  pResult[i] = 0;
  for (j=0; j<Size; j++)
    pResult[i] += pMatrix[i*Size+j]*pVector[j];
}
```

To calculate the time of one operation execution τ we measured the time spent on performing the algorithm for small object size, when matrix and vectors can fit in cache L1. We divide this time by the number of performed operations and get the time of one operation execution $\tau = 3{,}78$ nsec.

Table 1. Comparison of the experimental and theoretical execution time of the matrix-vector multiplication algorithm

Matrix Size	Experimental Time	Theoretical Time	Relative Error
2000	0,0303	0,0304	0,0021
4000	0,1222	0,1217	0,0036
6000	0,2748	0,2740	0,0029
8000	0,4894	0,4872	0,0044
10000	0,7637	0,7611	0,0034

In current version of ParaLab the simplier model for estimating the time of local computation is realized. This model only uses the number of operations and the time of one operation execution τ. We plan to implement the described approach to local computations time estimation in the next version of ParaLab.

3.2 Model for Data Passing Operations Execution

The time necessary for transmitting data between the processors defines *the communication overhead* of the parallel algorithm execution in a multiprocessor system. The basic set of parameters, which can help to evaluate the data transmission time, consists of the following values:

- **initializing time** (α) characterizes the duration of preparing the message for transmission, the search of the route in the network etc.;
- **control data transmission time** (t_c) between two neighboring processors (i.e. the processors, connected by a physical data transmission channel); to control data we may refer the message header, the error detection data block etc.;
- **transmission time of one data byte along a data transmission channel** ($1/\beta$); the duration of this transmission is defined by the communication channel bandwidth.

Let's consider *store-and-forward routing (SFR)*. In case of this approach the processor, which contains a message for transmission, gets all the amount of data ready for transmission, defines the processor, which should receive the data, and initializes the operation of data transmission. The processor, to which the message has been sent, first receives all the transmitted data and only then begins to send the received message further along the route. The time of data transmission t_{comm} for the method of transmitting the message of m bytes along the route of length l is defined by the expression:

$$t_{comm} = \alpha + (t_c + \frac{m}{\beta})l \tag{4}$$

If the messages are long enough, the control data transmission time may be neglected, and the expression for data transmission time may be written in a simplified way:

$$t_{comm} = \alpha + \frac{m}{\beta}l \tag{5}$$

Let's consider *cut-through routing (CTR)*, when the receiving processor may send the data further along the route immediately after receiving the current packet without waiting for the termination of the whole message data transmission. The data transmission time in case of packet communication method will be defined by the following expression:

$$t_{comm} = \alpha + \frac{m}{\beta} + t_c l \tag{6}$$

If we compare the obtained expressions, it is possible to notice that in the majority of cases the packet communication leads to faster data transmission. Besides, this approach decreases the need for memory for storing the transmitted data.

3.3 The Data Passing Operations in Multiprocessor and Multicore Architectures

As it was previously mentioned, in ParaLab the computational system consists of computational nodes, the network links between them are determined by the topology (farm, ring, etc.). Every node has one or more processors, every processor consists of one or more cores. We assume that the internal links between cores (busses) in frame of one node form the full graph topology.

To make the time estimation model easier we assume that the computations and data passing operations cannot overlap, which means that the computations stop when the cores are performing the data transmission, and vice versa.

Every collective data passing operation between cores can be divided into 3 stages:

1. Data transmission between cores in frames of one computational node and sending the data into the external network (via network adapters),
2. Data transmission between different computational nodes through the local network,
3. Receiving the data from the network adapter by the different cores in frames of one computational node.

To calculate the final time of the communication operation we only take into account the time of the second stage (passing the data through local network). The time spent on transmitting the data through the bus is 3 to 4 degrees less than that.

3.4 An Example of Computational Experiment Time Estimation

Let's consider the complexity of the parallel algorithm for matrix-vector multiplication based on rowwise matrix decomposition. Every core performs the multiplication of the matrix stripe by the vector, each stripe has n/p rows, where n is the size of the matrix and p is number of cores. One scalar product of the matrix row and a vector involves n multiplications and $(n\text{-}1)$ additions. Let's assume that the multiplication and addition have the same duration τ. Besides, let us assume that the computer system is homogeneous, i.e. all the processors of the system have the same performance. With regard to the introduced assumptions, the computation time of the parallel algorithm is:

$$T_p(calc) = \lceil n/p \rceil \cdot (2n-1) \cdot \tau \tag{7}$$

The 'all gather' operation is used to put the result vector on all the processes of the parallel program. This operation can be performed in $\lceil log_2 p \rceil$ iterations. At the first iteration the interacting pairs of processors exchange messages of size $w\lceil n/p \rceil$ byte (w is the size of one element of the vector in bytes). At the second iteration the size becomes doubled and is equal to $2w\lceil n/p \rceil$ etc. As a result, the all gather operation execution time when the Hockney [2] model is used can be represented as:

$$T_p(comm) = \sum_{i=1}^{\lceil log_2 p \rceil} \left(\alpha + \frac{2^{i-1} w \lceil n/p \rceil}{\beta} \right) = \alpha \lceil log_2 p \rceil + \frac{w \left\lceil \frac{n}{p} \right\rceil \left(2^{\lceil log_2 p \rceil} - 1\right)}{\beta} \tag{8}$$

where α is the latency of data communication network, β is the network bandwidth. Thus, the total time of parallel algorithm execution is

$$T_p = \left(\frac{n}{p}\right) \cdot (2n-1) \cdot \tau + \alpha \cdot log_2 p + w \cdot \left(\frac{n}{p}\right) \cdot (p-1)/\beta \tag{9}$$

(to simplify the expression (8) it was assumed that the values n/p and $log_2 p$ are whole numbers).

Let us analyze the results of the computational experiments carried out in order to estimate the efficiency of the discussed parallel algorithm. The obtained results will

be used for the comparison of the theoretical estimations and experimental values of the computation time. The experiments were carried out on the computational cluster on the basis of the processors Intel XEON 4 EM64T, 3000 Mhz and the network Gigabit Ethernet under OS Microsoft Windows Server 2003 Standard x64 Edition. The value of τ was equal to 1.93 nsec. The value of latency α and bandwidth β are correspondingly 47 msec and 53.29 Mbyte/sec. All the computations were performed over the numerical values of the double type, i.e. the value w is equal to 8 bytes.

The comparison of the experiment execution time T_p^* and the theoretical time T_p calculated in accordance with the expression (9) is shown in Table 2.

Table 2. The comparison of the experimental and theoretical execution time for parallel algorithm of matrix-vector multiplication based on rowwise matrix decomposition

Matrix Size	2 processors		4 processors		8 processors	
	T_p	T_p^*	T_p	T_p^*	T_p	T_p^*
1000	0,0069	0,0021	0,0108	0,0017	0,0152	0,0175
2000	0,0132	0,0084	0,014	0,0047	0,0169	0,0032
3000	0,0235	0,0185	0,0193	0,0097	0,0196	0,0059
4000	0,0379	0,0381	0,0265	0,0188	0,0233	0,0244
5000	0,0565	0,0574	0,0359	0,0314	0,028	0,015

4 Conclusion

The Parallel Laboratory software system (ParaLab) provides the possibility of carrying out computational experiments for studying and investigating the parallel algorithms of solving complicated computational problems. The system may be used for organizing a set of laboratory works on various courses in the area of parallel programming. This laboratory works will allow the learners to do the following:

- *To model multiprocessor systems* with various data communication network topologies,
- *To obtain the visual presentations of the computational processes and data communication operations* which take place in case of parallel solving various problems,
- *To construct the efficiency estimations* of the parallel methods to be studied.

In general, ParaLab is the integrated environment for studying and investigating the parallel algorithms of solving complicated computational problems. A wide set of available means to visualize the process of carrying out an experiment and to analyze the obtained results allows to study the parallel method efficiency on various computer systems, to make conclusions concerning the scalability of the algorithms and to determine the possible parallel computation speedup.

The processes of study and research realized by ParaLab are aimed at mastering the fundamentals of parallel computation theory. They allow the leaners to form the basic concepts of the models and methods of parallel computations through observation, comparison and analysis of various visual graphic forms demonstrated in the course of the experiment execution.

For those who only start to study the problem of parallel computations, ParaLab is very useful, as it allows them to master the parallel programming methods. Experienced users may use the system in order to estimate the efficiency of new parallel algorithms, which are being developed.

References

1. Foster, I.: Designing and Building Parallel Programs: Concepts and Tools for Software Engineering. Addison-Wesley, Reading (1995)
2. Hockney, R.W., Jesshope, C.R.: Parallel Computers 2. Architecture, Programming and Algorithms. Adam Hilger, Bristol (1988)
3. Kumar, V., Grama, A., Gupta, A., Karypis, G.: Introduction to Parallel Computing. The Benjamin/Cummings Publishing Company, Inc. (1994)
4. Quinn, M.J.: Parallel Programming in C with MPI and OpenMP. McGraw-Hill, New York (2004)
5. Buyya, R.: High Performance Cluster Computing, vol.1: Architectures and Systems, vol. 2: Programming and Applications. Prentice Hall PTR, Prentice-Hall Inc., Englewood Cliffs (1999)
6. Xu, Z., Hwang, K.: Scalable Parallel Computing Technology, Architecture, Programming. McGraw-Hill, Boston (1998)
7. Voevodin, V.V., Voevodin, V. V.: Parallel Computations, BHV, Saint-Petersburg (2002)
8. Gergel, V.P.: Theory and Practice of Parallel Computations. BINOM (2007)
9. Korneev, V.V.: Parallel Computational Systems, Knowledge, Moscow (1999)
10. Tanenbaum, E.: Computer Architecture, Piter, Saint-Petersburg (2002)