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

Lectures 5,6. Parallel Methods for Matrix-Vector Multiplication for Systems with Shared Memory

Nizhni Novgorod

2014

Lectures 5, 6_. Parallel Methods for Matrix-Vector Multiplication for Systems with Shared Memory

Matrices and matrix operations are widely used in mathematical modeling of various processes, phenomena and systems. Matrix calculations are the basis of many scientific and engineering calculations. Computational mathematics, physics, economics are only some of the areas of their application.

As the efficiency of carrying out matrix computations is highly important many standard software libraries contain procedures for various matrix operations. The amount of software for matrix processing is constantly increasing. New efficient storage structures for special type matrix (triangle, banded, sparse etc.) are being created. Highly efficient machine-dependent algorithm implementations are being developed. The theoretical research into searching faster matrix calculation method is being carried out.

Being highly time consuming, matrix computations are the classical area of applying parallel computations. On the one hand, the use of highly efficient multiprocessor systems makes possible to substantially increase the complexity of the problem solved. On the other hand, matrix operations, due to their rather simple formulation, give a nice opportunity to demonstrate various techniques and methods of parallel programming.

Let us assume that the matrices, we are considering, are dense, i.e. the number of zero elements in them is insignificant in comparison to the general number of matrix elements.

The repetition of the same computational operations for different matrix elements is typical of different matrix calculation methods. In this case we can say that there exist *data parallelism*. As a result, the problem to parallelize matrix operations can be reduced in most cases to matrix distributing among the processors of the computer system. The choice of matrix distribution method determines the use of the definite parallel computation method. The availability of various data distribution schemes generates a range of parallel algorithms of matrix computations.

The most general and the most widely used matrix distribution methods consist in partitioning data into *stripes* (vertically and horizontally) or rectangular fragments (*blocks*).

1. Block-striped matrix partitioning. In case of block-striped partitioning each processor is assigned a certain subset of matrix rows (*rowwise* or *horizontal partitioning*) or matrix columns (*columnwise* or *vertical partitioning*) (Figure 5.1). Rows and columns are in most cases subdivided into stripes on a continuous sequential basis. In case of such approach, in rowwise decomposition (see Figure 5.1), for instance, matrix A is represented as follows:

$$A = (A_0, A_1, \ldots, A_{p-1})^T,\ A_i = (a_{i_0}, a_{i_1}, \ldots, a_{i_{k-1}}),\ i_j = ik + j,\ 0 \le j < k,\ k = m / p\ ,$$

where $a_i = (a_{i1}, a_{i2}, \ldots a_{in}), 0 \le i < m,$ is *i-th* row of matrix A (it is assumed, that the number of rows *m* is divisible by the number of processors *p* without a remainder, i.e. $m = k \cdot p$). Data partitioning on the continuous basis is used in all matrix and matrix-vector multiplication algorithms, which are considered in this and the following sections.

Another possible approach to forming rows is the use of a certain row or column alternation (*cyclic*) scheme. As a rule, the number of processors *p* is used as an alternation parameter. In this case the horizontal partitioning of matrix A looks as follows:

$$A = (A_0, A_1, ..., A_{p-1})^T, A_i = (a_{i_0}, a_{i_1}, ..., a_{i_{k-1}}), i_j = i + jp, 0 \le j < k, k = m / p .$$

2. Checkerboard Block Matrix Partitioning. In this case the matrix is subdivided into rectangular sets of elements. As a rule, it is being done on a continuous basis. Let the number of processors be $p = s \cdot q$, the number of matrix rows is divisible by *s*, the number of columns is divisible by *q*, i.e. $m = k \cdot s$ and $n = l \cdot q$. Then the matrix *A* may be represented as follows:

$$A = \begin{pmatrix} A_{00} & A_{02} & \ldots A_{0q-1} \\ & \ldots & \\ A_{s-11} & A_{s-12} & \ldots A_{s-1q-1} \end{pmatrix},$$

where A_{ij} - is a matrix block, which consists of the elements:

$$A_{ij} = \begin{pmatrix} a_{i_0 j_0} & a_{i_0 j_1} & \ldots a_{i_0 j_{l-1}} \\ & \ldots & \\ a_{i_{k-1} j_0} & a_{i_{k-1} j_1} & a_{i_{k-1} j_{l-1}} \end{pmatrix}, i_v = ik + v, 0 \le v < k, k = m / s, \; j_u = jl + u, 0 \le u \le l, l = n / q .$$

In case of this approach it is advisable that a computer system have a physical or at least a logical processor grid topology of *s* rows and *q* columns. Then, for data distribution on a continuous basis the processors neighboring in grid structure will process adjoining matrix blocks. It should be noted however that cyclic alteration of rows and columns can be also used for the checkerboard block scheme.

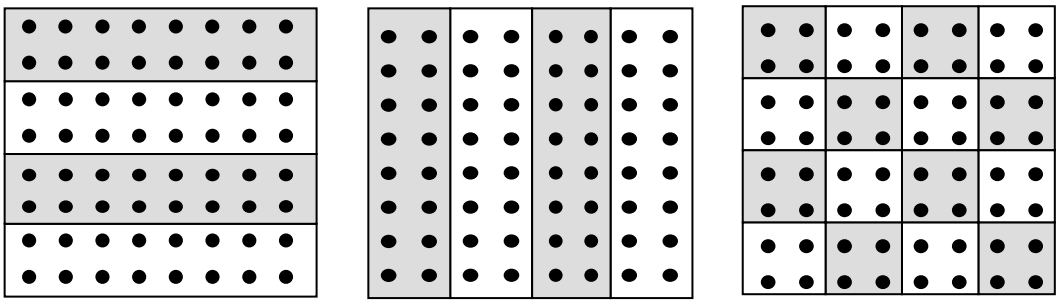

Figure 5.1 Most widely used matrix decomposition schemes

In this chapter three parallel algorithms are considered for square matrix multiplication by a vector. Each approach is based on different types of given data (matrix elements and vector) distribution among the processors. The data distribution type changes the processor interaction scheme. Therefore, each method considered here differs from the others significantly.

The lecture contains a detailed description of two possible matrix-vector multiplication algorithms using the mentioned ways of matrix decomposition. The first algorithm is based on rowwise matrix allocation to threads while the second one uses columwise matrix decomposition. Each algorithm is represented subject to the general scheme of parallel method development: first, basic subtasks are determined and then subtask information dependencies are identified followed by discussion of subtask scalability and their distribution among computational elements. In the end, efficiency of parallel computations are analyzed and experimental results are listed for each algorithm. For all described parallel matrix-vector multiplication algorithms, possible software implementations are proposed.

The resulting speedup and efficiency values show that all effective ways of data decomposition lead to uniform distribution of computational load while differences pertain only to complexity of the thread communication activities. In this respect, it will be interesting to see how the way of data decomposition influences the character of parallel program sections, identify basic differences in the way the threads use shared resources and operations required for access synchronization.

See the general diagram in Fig. 5.2 for speedup values obtained in the course of computational experiments for all the described algorithms. As one can see, the parallel matrix-vector multiplication algorithm in case of rowwise matrix decomposition has some advantage for speedup.

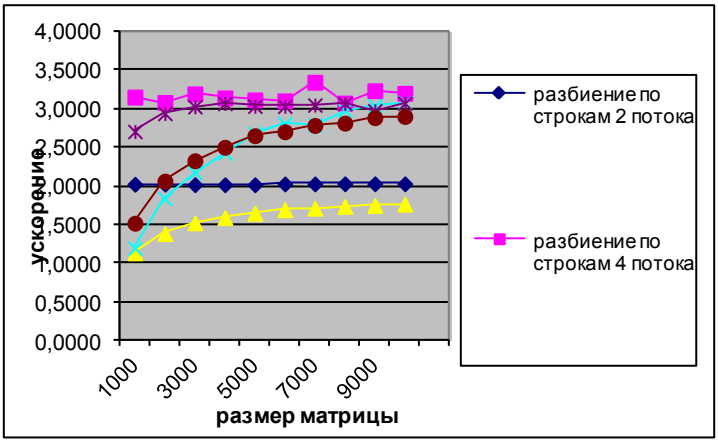

Fig. 5.2. Matrix multiplication speedup for the described parallel algorithms based on the results of computational experiments

Test questions

1. What are the main ways to distribute a matrix between threads?
2. What is the statement of the matrix-vector multiplication problem?
3. What is the computational complexity of a sequential matrix-vector multiplication algorithm?
4. What approaches can be proposed for development of parallel matrix-vector multiplication algorithms?
5. Give general schemes of the described matrix-vector multiplication.
6. Analyze and obtain efficiency parameters for one of the described algorithms.
7. Which of the described matrix-vector multiplication algorithms has the best speedup and efficiency?
8. Can data decomposition cycling influence runtime of each of the described algorithms?
9. What information communications are carried out for the algorithms in case of the block-striped data decomposition? In what way are the operations required for preparation of parallel program section and synchronization of access to common resources different for row-wise and columnwise matrix decomposition?
10. What information communications are carried out for the checkerboard matrix-vector multiplication algorithm?
11. What OpenMP tools and functions of the corresponding library proved to be necessary for program implementation of the algorithms?

Practice

1. Implement the parallel algorithm based on vertical block-striped matrix decomposition. Make theoretical runtime estimates for this algorithm taking into account your computer system parameters. Perform computational experiments. Compare actual experimental results with theoretical estimates.
2. Implement the parallel algorithm based on checkerboard block matrix decomposition. Make theoretical runtime estimates for this algorithm taking into account your computer system parameters. Perform computational experiments. Compare actual experimental results with theoretical estimates.

Reference

1. **Dongarra, J.J., Duff, L.S., Sorensen, D.C., Vorst, H.A.V. (1999).** Numerical Linear Algebra for High Performance Computers (Software, Environments, Tools). Soc for Industrial & Applied Math/
2. **Blackford,** L. S., Choi, J., Cleary, A., D'Azevedo, E., Demmel, J., Dhillon, I., Dongarra, J. J., Hammarling, S., Henry, G., Petitet, A., Stanley, D. Walker, R.C. Whaley, K. (1997). Scalapack Users' Guide (Software, Environments, Tools). Soc for Industrial & Applied Math.
3. **Foster**, I. (1995). Designing and Building Parallel Programs: Concepts and Tools for Software Engineering. Reading, MA: Addison-Wesley.