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”

TBB-Based Parallel Programming

Practice 1. Parallel Matrix-Vector Multiplication

Nizhni Novgorod

2014

Practice 1. Parallel Matrix-Vector Multiplication

Objectives

The purpose of this practice is to demonstrate parallel algorithms of linear algebra as illustrated by the matrix-vector multiplication problem.

Abstract

This practice formulates the matrix-vector multiplication problem. It features a sequential algorithm implementation. The practice also reviews variations of parallel algorithm implementation and offers a parallel_for-based algorithm implementation.

Guidelines

Having an easy definition, the matrix-vector multiplication problem is a classical problem that helps mastering development, debugging and optimization skills related to sequential and parallel programming. Such a wide application range is explained, first of all, by vast possibilities to demonstrate standard approaches to computation process engineering and efficient implementation of these approaches by means of programming languages. This practice deals with one of the possible approaches of solving this problem, its C++ sequential implementation and the TBB-based parallelization principles as illustrated by this approach. This fill affect only those TBB functions that are related to loop parallelization.

Multiplication of the matrix *A* sized *m x n* by the vector *b* consisting of *n* elements results in the vector *c* sized *m* with each i^{th} element resulting from scalar multiplication of the matrix *A* i^{th} row (let us call this row a_i) and the vector *b*.

$$c_i = (a_i, b) = \sum_{j=1}^{n} a_{ij} b_j \ , 1 \le i \le m$$

Therefore, obtaining the resulting vector c involves repetition of m operations to multiply the matrix *A* rows by the vector *b*. Each such operation includes multiplying a matrix row by the vector *b* (*n* operations) followed by summing the resulting products (*n-1* operations). The total number of scalar operations required is $T_1 = m \cdot (2n - 1)$.

A general scheme of the subproblem interaction during computations in case of striped data decomposition is shown in Fig. 1.

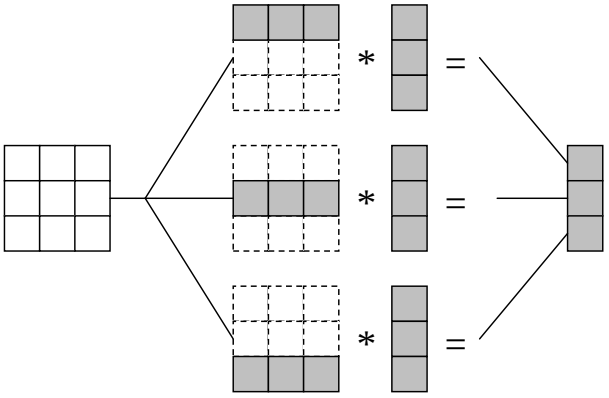

Fig. 1 Computing by means of a parallel matrix-vector multiplication algorithmbased on rowwise matrix partitioning

In the course of multiplication of a dense matrix partitioned into rows or columns by a vector, the number of computing operations to obtain the scalar product is the same for all basic subtasks. This enables basic subtask merger in a situation when the number of computing elements *p* is less than the number of basic subproblems *m (p<m),* so that each computing element solves several problems that correspond to a continuous sequence of matrix rows.

Recommendations for Students

Cormen, Leiserson, Rivest, Stein (2009) is a recommended introduction to algorithms.

Quinn (2004) is also recommended as a description of typical problems of parallel programming.

Practice

1. Adapt the existing implementation to a non-square matrix.
2. Develop a program for multiplication of square matrices.
3. Develop a rectangular matrix multiplication program.

Test questions

1. What is the computational complexity of a matrix-vector multiplication algorithm (n is the matrix size)?
   a. O(n).
   b. (+) O(n2).
   c. O(n3).
2. What are the main ways to allocate effort to threads?
   a. (+) Rowwise
   b. (+) Columnwise
   c. (+) Blockwise
   d. By matrix QR and LU-factorization base
3. What is the total number of scalar operations of a matrix-vector multiplication algorithm (m is the number of matrix rows and n is the number of vector columns)?
   a. m•(n – 1).
   b. (+) m•(2n – 1).

c. m2•n.

d. m•n2.

e. 2m•(2n – 1).

4. What grainsize value is the most suitable for the matrix-vector multiplication problem for a 4-core computer (m is the number of matrix rows)?

a. 4.

b. (+) m/50.

c. m/5000.

5. What is the best for implementation of the matrix-vector multiplication algorithm?

a. (+) parallel_for

b. parallel_reduce

c. task

6. Pointers to the matrix and multiplied vector:

a. Must be stored in functor fields

b. (+) Must be stored in constant functor fields

c. Must be stored in global variables.

d. Must be passed to operator().

7. Pointer to the resulting vector:

a. Must be stored in functor fields

b. (+) Must be stored in constant functor fields

c. Must be stored in global variables.

d. Must be passed to operator().

8. In its fields, the functor must store:

a. (+) Pointer to the resulting vector.

b. (+) Pointers to the matrix and multiplied vector:

c. Number of the currently processed element.

d. Number of matrix rows

e. (+) Number of matrix columns

References

1. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein C. (2009). Introduction to Algorithms, 3rd Edition. – The MIT Press.

2. Kumar V., Grama, A., Gupta, A., Karypis, G. (1994). Introduction to Parallel Computing. - The Benjamin/Cummings Publishing Company, Inc. (2nd edn., 2003)

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

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