Recursive Matrix Algorithms, Distributed Dynamic Control, Scaling, Stability

Gennadi Malaschonok
Faculty of Informatics
National University of Kyiv-Mohyla Academy
Kyiv, Ukraine
malaschonok@gmail.com

***Abstract*—The report is devoted to the concept of creating block-recursive matrix algorithms for computing on a supercomputer with distributed memory and dynamic decentralized control.**

***Index Terms*—lock-recursive matrix algorithms, distributed dynamic control, distributed memory, scaling, stabilitylock-recursive matrix algorithms, distributed dynamic control, distributed memory, scaling, stabilityb**

I. Introduction

Appearance of the supercomputer system with hundreds of thousands of cores poses many new problems for specialists in the field of parallel computing. The three main ones are uniform load of equipment, the presence of control over the growth of the error of numbers during calculations and the presence of protection against possible physical failures of individual processors.

In the paper [1], the authors presented a new task insertion extension for PaRSEC, Dynamic Task Discovery (DTD), supporting shared and distributed memory environments. They compare two programming paradigms: Parameterized Task Graph (PTG) and Dynamic Task Discovery (DTD). The result shows good scalability and comparable result to PTG in most cases and, where comparable benchmarks exist, consistently better performance compared to other runtime.

We suggest using another dynamic control scheme for a parallel computing process, which is much simpler than DTD and does not allow to control the parallel execution of an arbitrary algorithm. It can be used only for block-recursive algorithms. In such algorithms, independent separate subtasks operations apply to blocks, so it is easy to organize decentralized control of the entire computational process.

The second problem is the accumulation of errors during calculations. The larger the matrix size, the more error can accumulate.

Let a set of matrices be given and it is required to calculate some new matrices, vectors or scalars. All source numbers are rational numbers due to the fact that the memory has a finite size. If your algorithm uses only rational operations, then you have the opportunity to get an exact answer with respect to the input data.

If the approximate calculations are used, then the calculations error increases with the number of operations. Consequently, with the growth of matrix sizes, there comes a moment when the error exceeds the allowed limits. For example, in the Gauss algorithm, errors can exceed the exact solution already for matrices of order 10 if these matrices are ill-conditioned. Unfortunately, for every well-conditioned matrix, this boundary also has a well-defined value. And what should be done if the size of the matrix exceeds this limit value?

Then you have to change the computational paradigm. For example, you can exchange accuracy for time, but you have many different possibilities to do this.

The question is, what should be the new computational paradigm? It may vary depending on the type of algorithm. Let's consider these types.

II. Three classes of matrix algorithms

All matrix algorithms are divided into three separate classes: (MA_1) the rational direct matrix algorithms, (MA_2) the irrational direct matrix algorithms, (MA_3) the iterative matrix algorithms.

The first class (MA_1) contains algorithms that use only four arithmetic operations. As a result, only rational functions can be computed. This class includes an algorithm for solving systems of linear equations, calculating the inverse matrix, a determinant, a similar three-diagonal matrix, a characteristic polynomial, a generalized inverse matrix, a kernel of a linear operator, LU, LEU and LDU decompositions, Bruhat decomposition and so on.

The second (MA_2) class consists of all direct methods that did not fall into the first class. Elements of matrices that are obtained as a result of the application of these methods cannot be obtained in the form of rational functions. This class includes algorithms for QR-decomposition of matrices, calculations of a similar two-diagonal matrix, and others.

The third class (MA_3) consists of all remaining algorithms, in which iterative methods are used. For example, algorithms for calculating eigenvalues and eigenvectors of a matrix and algorithms for SVD decomposition fall into this class when the rank of the matrix is greater than four.

Here you can see a complete analogy with algorithms for solving algebraic equations. Algorithms for solving algebraic equations can be divided into the same three classes. The first class contains algorithms for solving linear equations. The second class consists of direct algorithms for solving equations

of the second, third and fourth degrees. And the third class consists of iterative algorithms for solving algebraic equations. Such algorithms allow finding solutions to equations of degree five and higher.

Each of these classes uses its own special matrix algorithms. Accordingly, each of these classes requires its own methods of creating matrix algorithms for large matrices and for supercomputers with distributed memory.

III. MA1-ALGORITHMS

We will assume that all matrices are square and have 2^k rows and columns. If the matrix has other sizes, then it can be added to such a square matrix with zero or unit blocks.

A. Recursive standard and Strassen's matrix multiplication

Recursive algorithm for standard matrix multiplication is based on the equations

$$\begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix} \times \begin{pmatrix} B_0 & B_1 \\ B_2 & B_3 \end{pmatrix} + \begin{pmatrix} C_0 & C_1 \\ C_2 & C_3 \end{pmatrix} = \begin{pmatrix} D_0 & D_1 \\ D_2 & D_3 \end{pmatrix}$$

$D_0 = A_0B_0 + A_1B_2 + C_0, D_1 = A_0B_1 + A_1B_3 + C_1, D_2 = A_2B_0 + A_3B_2 + C_2, D_3 = A_2B_1 + A_3B_3 + C_3$. Number of operations for this algorithm is $\sim n^3$.

The Strassen multiplication algorithm [2] is also a block recursive algorithm. The number of operations for this algorithm is $\sim n^{\log_2 7}$. There exists a boundary with respect to the density of the matrix, which separates the region of applicability of the Strassen multiplication. (see details in [3]).

B. Recursive inversion of triangular matrix

If $\mathcal{A} = \begin{pmatrix} A & 0 \\ B & C \end{pmatrix}$ is invertible triangular matrix of order 2^k and $\det(\mathcal{A}) \neq 0$ then

$$\mathcal{A}^{-1} = \begin{pmatrix} A^{-1} & 0 \\ -C^{-1}BA^{-1} & C^{-1} \end{pmatrix}.$$

C. Recursive Cholesky decomposition

Let $\mathcal{A} = \begin{pmatrix} A_1 & A_2 \\ A_2^T & A_3 \end{pmatrix}$ be a positive definite symmetric matrix and $H = \begin{pmatrix} B & 0 \\ C & D \end{pmatrix}$ be a low triangle matrix with the property $\mathcal{A} = HH^T$. The mapping

$$Chol(\mathcal{A}) = (H, H^{-1})$$

is called an *Cholesky decomposition*. It is easy to see that the recursive algorithm of Cholesky decomposition has the following form. Let $Chol(A_1) = (B, B^{-1})$. Then we can compute

$$C = A_2^T(B^{-1}) \quad \text{and} \quad F = A_3 - CC^T$$

Let $Chol(F) = (D, D^{-1})$. Then $H = \begin{pmatrix} B & 0 \\ C & D \end{pmatrix}$ and $H^{-1} = \begin{pmatrix} B^{-1} & 0 \\ -D^{-1}CB^{-1} & D^{-1} \end{pmatrix}$.

D. Recursive Strassen's matrix inversion

If $\mathcal{A} = \begin{pmatrix} A_0 & A_1 \\ A_2 & A_3 \end{pmatrix}$, $\det(\mathcal{A}) \neq 0$ and $\det(A_0) \neq 0$ then the inverse matrix can be calculated as follows:

$$\mathcal{A}^{-1} = \begin{pmatrix} \boldsymbol{I} & -A_0^{-1}A_1 \\ 0 & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} \boldsymbol{I} & 0 \\ 0 & (A_3 - A_2A_0^{-1}A_1)^{-1} \end{pmatrix} \times \begin{pmatrix} \boldsymbol{I} & 0 \\ -A_2 & \boldsymbol{I} \end{pmatrix} \begin{pmatrix} A_0^{-1} & 0 \\ 0 & \boldsymbol{I} \end{pmatrix} = \begin{pmatrix} M_6 & M_1M_4 \\ M_5 & M_4 \end{pmatrix}$$

We have denoted here $M_0 = -A_0^{-1}$, $M_1 = M_0A_1$, $M_2 = A_2M_0$, $M_3 = M_2A_1$, $M_4 = (A_3 + M_3)^{-1}$, $M_5 = -M_4M_2$, $M_6 = M_1M_5 - M_0$. (see in [2]).

E. Other recursive matrix algorithms of MA1-class

You can find many other recursive matrix algorithms of this class in the papers [4]- [11]. These are such algorithms as computation of the adjoint matrix, kernel and matrix determinant, computation of the generalized Bruhat decomposition in fields and in commutative domains, LEU and LDU triangular decomposition of matrices. New applications of these algorithms were preposed in [12] and [13].

F. Conclusion

All NA_1 class algorithms have a complexity of $\sim n^3$ (or $\sim n^\beta$) in operations on matrix elements using standard matrix multiplication (or fast matrix multiplication with n^β operations). For numerical matrices, one can obtain exact solutions by spending another n^2 (or n^α, or n) bit operations for standard multiplication of numbers (or fast multiplication of numbers (with complexity n^α), or the use of finite fields).

In all these algorithms, we obtain an exact solution and the question of the accumulation of error does not arise here.

As we can see, many block recursive algorithms are already known in the class MA_1. However, in the next class MA_2, we know only one such algorithm. This is Schonhage algorithm for the QR-decomposition of a matrix [14].

In the next section, we propose another way of presenting the algorithm [14] and we calculate the exact number of operations.

IV. MA2-CLASS: QR DECOMPOSITION

Let A be a matrix over a field. It is required to find the upper triangular matrix R and the orthogonal Q matrix such that $A = QR$. For definiteness, we will consider an algorithm applied to a square matrix A over a field of real numbers.

Consider the case of a 2×2 matrix. The desired decomposition $A = QR$ has the form:

$$\begin{pmatrix} \alpha & \beta \\ \gamma & \delta \end{pmatrix} = \begin{pmatrix} c & -s \\ s & c \end{pmatrix} \begin{pmatrix} a & b \\ 0 & d \end{pmatrix},$$

where the numbers s and c satisfy the equation $s^2 + c^2 = 1$. If $\gamma = 0$ then we can set $c = 1,\ s = 0$. If $\gamma \neq 0$, then we get $\Delta = \alpha^2 + \gamma^2 > 0$, $c = \alpha/\sqrt{\Delta}$, $\quad s = \gamma/\sqrt{\Delta}$. We denote such a matrix Q by $g_{\alpha,\gamma}$.

Let the matrix A be given, its elements (i, j) and $(i+1, j)$ be α and γ, and all the elements to the left of them be zero: $\forall(s < j) : (a_{i,s} = 0)\ \&\ (a_{i+1,s} = 0)$.

Let $G_{i,j} = \mathbf{diag}(I_{i-1}, g_{\alpha,\gamma}, I_{n-i-1})$. These matrices are called Givens matrices. Then the matrix $G_{i,j}A$ differs from A only in two rows i and $i+1$, but all the elements to the left of the column j remain zero, and in the column j in $i+1$ line will be 0.

This property of the Givens matrix allows us to formulate such an algorithm

(1). First we reset the elements under the diagonal in the left column:

$$A_1 = G_{1,1}G_{2,1}...G_{n-2,1}G_{n-1,1}A$$

(2). Then we reset the elements that are under the diagonal in the second column:

$$A_2 = G_{2,2}G_{3,2}...G_{n-2,2}G_{n-1,2}A_1$$

(k). Denote $G_{(k)} = G_{k,k}G_{k-1,k}...G_{n-2,k}G_{n-1,k}$, $k = 1, 2, .., n-1$. Then, to calculate the elements of the k th column, we need to obtain the product of matrices

$$A_k = G_{(k)}A_{k-1}.$$

(n-1). At the end of the calculation, the element in the $n-1$ column will be reset: $A_{n-1} = G_{(n-1)}A_{n-2} = G_{n-1,n-1}A_{n-2}$.

A. QR_G decomposition

Let a matrix M of size $2n \times 2n$ be divided into four equal blocks: $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$. There are three stages in this algorithm.

(1). The first stage is the QR_G decomposition of the block C:

$$C = Q_1C_1,\ M_1 = \mathbf{diag}(I, Q_1)M = \begin{pmatrix} A & B \\ C_1 & D_1 \end{pmatrix}.$$

(2). The second stage is the cancellation of a parallelogram composed of two triangular blocks: the lower triangular part A^L of the block A and the upper triangular part C_1^U of the block C_1. Denote the upper triangular matrix A_1 and annihilating matrix Q_2:

$$Q_2 \begin{pmatrix} A \\ C_1 \end{pmatrix} = \begin{pmatrix} A_1 \\ 0 \end{pmatrix},\ M_2 = Q_2M_1 = \begin{pmatrix} A_1 & B_1 \\ 0 & D_2 \end{pmatrix}.$$

(3). The third stage is the QR_G decomposition of the D_2 block: $D_2 = Q_3D_3$.

$$R = \mathbf{diag}(I, Q_3)M_2 = \begin{pmatrix} A_1 & B_1 \\ 0 & D_3 \end{pmatrix}.$$

As a result, we get:

$$M = Q^TR,\quad Q = \mathbf{diag}(I, Q_3)Q_2\,\mathbf{diag}(I, Q_1).$$

Since the first and third stages are recursive calls of the QR_G procedures, it remains to describe the parallelogram cancellation procedure. Let's call it a QP decomposition.

B. QP-decomposition

Let the matrix $M = \begin{pmatrix} A \\ B^U \end{pmatrix}$ have dimensions $2n \times n$ and, at the same time, the lower unit B^U of size $n \times n$ has an upper triangular shape - all elements under its main diagonal are zero. We are looking for the factorization of the matrix $M = QP = Q\begin{pmatrix} A^U \\ 0 \end{pmatrix}$, with the orthogonal matrix Q.

It is required to annul all elements between the upper and lower diagonals of the M matrix, including the lower diagonal. It is easy to see that this can be done with Givens matrices. We will consistently perform columns invalidation by traversing column elements from bottom to top and traversing columns from left to right.

But we are interested in the block procedure. Since n is even, we can break the parallelogram formed by the diagonals into 4 parts using its two middle lines. We get 4 equal parallelograms. To cancel each of them, we will simply call the parallelogram cancellation procedure 4 times. We will perform the calculations in this order: the bottom left (P_{ld}), then we simultaneously cancel the top left (P_{lu}) and the bottom right (P_{rd}), and the last we will cancel the top right (P_{ru}). The corresponding orthogonal Givens matrices of size $n \times n$ are denoted Q_{ld}. Q_{lu}. Q_{rd} and Q_{ru}. Let

$$\bar{Q}_{ld} = \mathbf{diag}(I_{n/2}, Q_{ld}, I_{n/2}),\ \bar{Q}_{ru} = \mathbf{diag}(I_{n/2}, Q_{ru}, I_{n/2}),$$

As a result, we get:

$$Q = \bar{Q}_{ru}\,\mathbf{diag}(Q_{lu}, Q_{rd})\bar{Q}_{ld}$$

The number of multiplications of matrix blocks of size $n/2 \times n/2$ is 24. Hence the total number of operations: $Cp(2n) = 4Cp(n) + 24M(n/2)$.. Suppose that for multiplication of two matrices of size $n \times n$ you need γn^β operations and $n = 2^k$, then we get: $Cp(2^{k+1}) = 4Cp(2^k) + 24M(2^{k-1}) = 4^kCp(2^1) + 24\gamma\sum_{i=0}^{k-1} 4^{k-i-1}2^{i\beta} =$
$24\gamma(n^2/4)\frac{2^{k(\beta-2)}-1}{2^{(\beta-2)}-1} + 6n^2 = 6\gamma\frac{n^\beta - n^2}{2^\beta - 4} + 6n^2$

$$Cp(n) = \frac{6\gamma n^\beta}{2^\beta(2^\beta - 4)} + \frac{3n^2}{2}(1 - \frac{\gamma}{2^\beta - 4})$$

C. The complexity of QR_G decomposition algorithm

Let us estimate the number of operations $C(n)$ in this block-recursive decomposition algorithm, assuming that the complexity of the matrix multiplication is $M(n) = \gamma n^\beta$, the complexity of canceling the parallelogram is $Cp(n) = \alpha^\beta$, where α, β, γ are constants, $\alpha = \frac{6\gamma}{2^\beta(2^\beta - 4)}$ and $n = 2^k$: $C(n) = 2C(n/2) + Cp(n) + 6M(n/2) = 2C(2^{k-1}) + Cp(2^k) + 6M(2^{k-1}) =$

$$= \frac{\gamma 6(2^\beta - 3)(n^\beta - \frac{2n}{2^\beta})}{(2^\beta - 4)(2^\beta - 2)}$$

D. Conclusions

These two MA_2-class algorithms have a complexity of $\sim n^3$ (or $\sim n^\beta$) in operations on matrix elements using standard matrix multiplication (or fast matrix multiplication with $\sim n^\beta$ operations).

We cannot avoid rounding errors. Therefore, it is necessary to be able to control the calculation error by increasing the number of digits for storing numbers.

Control of calculation errors in MA_3-class requires special additional studies.

V. Dynamic algorithms

Dynamic matrix algorithms are based on matrix block-recursive algorithms. In such algorithms, the matrix is recursively divided into blocks. A block-recursive algorithm is again applied to each of the blocks. This happens as long as the blocks remain large enough. When the block size becomes small enough, the usual sequential algorithms are applied to the blocks. This limit for the size of a small block depends on the physical characteristics of the computing device and should be automatically adjusted to the specific equipment.

A. The dynamic algorithm has three stages

First stage. This is the initial construction of the connections tree for computational nodes. The large blocks are sent from the root node to a child along with lists of free nodes. From these child nodes, data is sent further, but already with smaller blocks and corresponding parts of the list of free nodes.

Second stage. It occurs when either all the free nodes have received their subtasks, or when the size of the blocks has decreased to a certain boundary, which is predetermined. The tree of connections is constructed and the calculation takes place on leaf vertices.

The third stage. At this stage, the results are returned from leaf vertices to the root vertex. The result of the main task is obtained at root vertex and the calculations are completed.

B. Automatic redistribution of subtasks

Dynamic control involves the automatic redistribution of subtasks from overloaded nodes to free nodes. For this purpose, a scheme is provided for transmitting information about free nodes and information about overloaded nodes. Both streams of information are transmitted along the tree towards the root vertex until they meet at a certain node. After this, the information about free vertices is redirected to the overloaded vertices.

The largest subtasks from the overloaded nodes are transmitted to the free nodes. And after completing the calculations, the result is returned to the node from which this subtask was obtained.

C. Protection scheme in case of a failure of a node

It also uses a very simple protection scheme in case of a failure of a node during calculations.

Let node 1 send a subtask S to node 2. Let node 2 fail and the failure message cames to node 1. Node 1 will mark this subtask S as unsolved and return it to the list of unsolved subtasks. All operations of transferring results from child nodes to node 2 are simply canceled. No other action is required. The computational process will continue on all other nodes without any changes.

This scheme was implemented in the Java programming language using the OpenMPI and MathPartner [15] packages, and its work was tested on the matrix multiplication and matrix inversion algorithms. A more detailed description of this scheme is presented in [16].

VI. Conclusion

We proposed a new classification of matrix computational algorithms, which decomposes all algorithms into three classes: rational, irrational and iterative. We discribed the new computational paradigm: using of the block-recursive matrix algorithms for creating parallel programs that are designed for supercomputers with distributed memory and dynamic decentralized control of the computational process. We have shown many examples of such algorithms. We proposed a dynamic decentralized computation control scheme.

References

[1] Reazul Hoque, Thomas Herault, George Bosilca, Jack Dongarra: Dynamic Task Discovery in PaRSEC- A data-flow task-based Runtime. Proc. ScalA17, Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems, November 12-17, 2017, Denver, CO, USA (2017)

[2] Strassen V. Gaussian Elimination is not optimal. Numerische Mathematik. V. 13, 354-356 (1969)

[3] Malaschonok G.I. , Valeev Yu D. and Lapaev A.O.: On the choice of a multiplication algorithm for polynomials and polynomial matrices. Journal of Mathematical Sciences. Volume 168, Number 3, pp. 398-416 (2009)

[4] Malaschonok G.I.: Effective Matrix Methods in Commutative Domains. Formal Power Series and Algebraic Combinatorics. Springer, Berlin pp. 506-517. (2000).

[5] Malaschonok G.I.: Matrix computational methods in commutative rings. Monograph. Tambov, Tambov University Publishing House (2002).

[6] Akritas A.G., Malaschonok G.I.: Computation of Adjoint Matrix, Computational Science, ICCS 2006, LNCS 3992, Springer, Berlin, pp. 486–489, (2006)

[7] Malaschonok G.: On computation of kernel of operator acting in a module. Tambov University Reports. Series: Natural and Technical Sciences, vol. 13, issue 1, pp. 129–131, (2008)

[8] Malaschonok G.I.: Fast Generalized Bruhat Decomposition. In: Ganzha, V.M., Mayr, E.W., Vorozhtsov, E.V. (eds.) 12th International Workshop on Computer Algebra in Scientific Computing (CASC 2010), LNCS 6244. Springer, Berlin Heidelberg, pp. 194-202 (2010)

[9] Malaschonok G.I.: On the fast generalized Bruhat decomposition in domains, Tambov University Reports, vol. 17, no. 2, 544-550 (2012).

[10] Malaschonok G.: Generalized Bruhat decomposition in commutative domains, Computer Algebra in Scientific Computing, CASC'2013, LNCS 8136, Springer, Heidelberg, 2013, pp. 231-242 (2013)

[11] Malaschonok G., Scherbinin A.: Triangular Decomposition of Matrices in a Domain, Computer Algebra in Scientific Computing, LNCS 9301, Springer, Switzerland, 2015, pp. 290-304 (2015)

[12] Bouman Niek J. and de Vreede Niels: New Protocols for Secure Linear Algebra: Pivoting-Free Elimination and Fast Block Recursive Matrix Decomposition. Cryptology ePrint Archive, Report 2018/703. Available at https://eprint.iacr.org/2018/703. (2018)

[13] Pernet C., Storjohann A.: Time and space efficient generators for quasiseparable matrices, Journal of Symbolic Computation, vol. 85, no. 2, pp. 224-246 (2018)

[14] Schönhage A.: Unitre Transformationen groer Matrizen, Numerische Mathematik 20, pp. 409-417 (1973)

[15] Malaschonok G.I.: MathPartner Computer Algebra. Programming and Computer Software, Vol. 43, No. 2, pp. 112-118 (2017)

[16] Malaschonok G.I., Sidko A.A.: Parallel computer algebra: a new scheme for controlling the parallelization of matrix recursive algorithms: Fifth International Conference on High Performance Computing (HPCUA 2018) October 22-23, 2018 in Kyiv, Ukraine, pp. 77-85 (2018)