Algebraic classification of matrix algorithms

Gennadi Malaschonok

Abstract. The widespread use of supercomputers for solving practical problems with numerical matrices of large sizes requires a revision of the matrix algorithms that are used today. The Galois classification for dividing all matrix algorithms into three classes is discussed. The special role of recursive matrix algorithms is emphasized. These algorithms provide a natural way to isolate independent large-block subproblems for parallel computations on a supercomputer.

1. Introduction

The theory of matrices and matrix algorithms that was created for the computing technology of past generations needs to be revised. This has been repeatedly stated by Jack Dongarra, who is rightfully considered the father of supercomputing. He calls for the search for a new computing paradigm. We can cite as an example [1].

We propose to consider several proposals that could form the basis of such a new paradigm.

Earlier we dealt with "small matrices" and computers with shared memory. Now we need to learn how to deal with (A) "large matrices" and (B) supercomputers that have distributed memory. Each individual processor of a supercomputer has memory available only to its cores. In addition, (C) in large computing clusters, while solving a problem, individual processors are likely to fail.

The main problem is the loss of the correct solution to the problem due to the accumulation of computational error. Overcoming this difficulty is possible in only one of two ways.

(I) You can increase the bit width of the machine word in which numbers are stored. For this purpose, you can, for example, use the Java BigDecimal class.

(II) It is possible to find an exact solution to the problem for some tasks. For this purpose, you can, for example, use the Java BigInteger class.

2. Galois classification of matrix algorithms

In connection with such a revision of matrix algorithms, it is necessary to take into account the algebraic features of the algorithms.

We will consider rational numbers (or their complexification) as input data. Then, as a result of calculations in the solution, one can obtain either rational numbers (Q) or numbers that are expressed in radicals (ER), or algebraic numbers, that are not expressed in radicals (AR).

Let's call these three classes of matrix algorithms MA_i $(i = 1, 2, 3)$. Or otherwise, we can talk about a Q, ER and AR classes of matrix algorithms. We propose to call it the Galois classification, since it has a direct connection with the calculation of the roots of a polynomial. Class MA_1 is similar to the class of linear algebraic equations and has only rational solutions. Class MA_2, is similar to algebraic equations of 2,3 and 4th degree and has solutions that are expressed in radicals. Class MA_3, is similar to equations above the fourth degree and has solutions that are not expressed in radicals.

Dividing the entire set of matrix problems into these three large groups is a reasonable first step in creating a modern theory of matrix algorithms.

Indeed, in the first class one can use a large arsenal of algebraic techniques, including transition to finite fields, rational reconstruction, or use the operation of exact integer division.

In the second class, such techniques are impossible. But here you can get a solution without using an iterative process. For example, to reduce the error, it is enough to switch to calculations with numbers, which have increased digit capacity.

In the third class of algorithms, there must be an iterative process in which a solution is achieved with the required accuracy.

3. Block-recursive matrix algorithms

The first non-trivial block-recursive matrix algorithm is Strassen's algorithm for matrix multiplication. In recent years, new block-recursive matrix algorithms have been obtained. The main feature of such algorithms, which is fundamentally important for supercomputer computing, is the ability to separate large independent subproblems.

A less important but very useful property of these algorithms is their lower computational complexity. It usually corresponds to the computational complexity of the multiplication algorithm. For example, when using Strassen's algorithm, the complexity of the entire recursive algorithm will be $\sim n^{log_2 7}$.

In recent years, many research groups have been intensively searching for special mechanisms for extracting large independent subproblems from the algorithm. Here are some of the task-based runtimes known today: OpenMP [2], StarPU [3], Legion [4], PaRSEC [5], OCR [6], HPX [7], SuperGlue[9], QUARK [10]. A framework for synthesizing distributed-memory parallel programs for block recursive

algorithms is presented in [8] and is illustrated by synthesizing programs for the FFT. It has been incorporated into the EXTENT system.

We continue to develop this area for creating an efficien runtime for supercomputer. Our DAP-technology [20] – [22] exploits the natural property of separating independent recursive subproblems in a recursive algorithm.

Today, such recursive algorithms are known in MA_1 class: multiplication, inversion, calculation of the adjoint matrix and determinant, calculation of the echelon form and kernel of the operator, calculation of the pseudoinverse matrix, generalized Moore-Penrose inverse, Bruhat decomposition, LEU and LDU expansions. In the MA_2 class, a recursive algorithm for QR-decomposition is known.

4. Algorithms in the commutative domain

Computer algebra studies algorithms in various universal algebras, including commutative domains.

As is known, a commutative domain can naturally be immersed in its own field of quotients. However, operations in the field of quotients, as a rule, are much more computationally complex. Even in cases where the solution lies in the field of quotients, an algorithm that does not use arithmetic in the field of quotients, but presents the numerators and denominators of the desired fractions, may have less complexity.

If, however, homomorphic images of the field of quotients into finite fields are used, and then a rational reconstruction of the solution is applied, then here, too, the computational complexity can be reduced. This is due to the fact that reconstruction of a separate numerator and denominator has less computational complexity than reconstruction of a fraction.

Sparse matrices are of particular interest. It is well known that the use of homomorphic mappings reduces the computational complexity for dense matrices. However, the picture may be different for sparse matrices. And for some types of sparse matrices, the use of a homomorphic mapping may not improve, but worsen the computational complexity.

Thus, we can distinguish a special subset of algorithms in class A. These are block-recursive algorithms for a commutative domain. They are the most promising for supercomputing.

5. Conclusion

We propose to revise matrix algorithms in connection with the requirements put forward by supercomputing, and to base it on the division into three classes: MA_1, MA_2, MA_3.

For each of these classes, one can further consider their own approaches to solving supercomputing problems.

We also note the special position occupied by block-recursive matrix algorithms for solving the problem of allocating large independent subtasks when computing on a supercomputer, which has memory distributed across separate processors.

Among block-recursive algorithms of MA_1 class, a special place is occupied by algorithms for commutative domains, which make it possible to further reduce the computational complexity of an algorithm.

References

[1] Dongarra J. *With Extrim Scale Computing the Rules Have Changed.* In Mathematical Software. ICMS 2016, 5th International Congress, Procdistributed memoryeedings (G.-M. Greuel, T. Koch, P. Paule, A. Sommese, eds.), Springer, LNCS, volume 9725, pp. 3-8, (2016)

[2] OpenMP 4.0 Complete Specifications. (2013). http://www.openmp.org/ wp-content/uploads/OpenMP4.0.0.pdf

[3] Emmanuel Agullo, Olivier Aumage, Mathieu Faverge, Nathalie Furmento, Florent Pruvost, Marc Sergent, and Samuel Thibault. 2014. Harnessing Supercomputers with a Sequential Task-based Runtime System. 13, 9 (2014), 1–14.

[4] Michael Bauer, Sean Treichler, Elliott Slaughter, and Alex Aiken. 2012. Legion: Expressing locality and independence with logical regions. In International Conference for High Performance Computing, Networking, Storage and Analysis, SC. https://doi.org/10.1109/SC.2012.71

[5] George Bosilca, Aurélien Bouteiller, Anthony Danalis, Mathieu Faverge, Thomas Hérault, and Jack Dongarra. 2013. PaRSEC: A programming paradigm exploiting heterogeneity for enhancing scalability. Computing in Science and Engineering 99 (2013), 1. https://doi.org/10.1109/MCSE.2013.98

[6] Jiri Dokulil, Martin Sandrieser, and Siegfried Benkner. 2016. Implementing the Open Community Runtime for Shared-Memory and Distributed-Memory Systems. Proceedings - 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2016 (2016), 364–368. https://doi.org/10.1109/PDP.2016.81

[7] T. Heller, H. Kaiser, and K. Iglberger. 2013. Application of the ParalleX execution model to stencil-based problems. Computer Science - Research and Development 28, 2-3 (2013), 253–261. https://doi.org/10.1007/s00450-012-0217-1

[8] S. K. S. Gupta, C.-H. Huang, P. Sadayappan, and R. W. Johnson. A framework for generating distributed-memory parallel programs for block recursive algorithms. J. Parallel and Distributed Computing, 34:137–153, 1996.

[9] Martin Tillenius. 2015. SuperGlue: A Shared Memory Framework Using Data Versioning for Dependency-Aware Task-Based Parallelization. SIAM Journal on Scientific Computing 37, 6 (2015), C617–C642. https://doi.org/10.1137/140989716

[10] Asim Yarkhan. 2012. Dynamic Task Execution on Shared and Distributed Memory Architectures. December (2012). http://trace.tennessee.edu/utk

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

[12] Malaschonok G. *Matrix computational methods in commutative rings.* Tambov, TSU, 213 p. (2002)

[13] Akritas A.G., Malaschonok G.I. *Computation of Adjoint Matrix.* Computational Science, ICCS 2006, LNCS 3992, Springer, Berlin, 486-489.(2006)

[14] Malaschonok G. *On computation of kernel of operator acting in a module* Vestnik Tambovskogo universiteta. Ser. Estestvennye i tekhnicheskie nauki [Tambov University Reports. Series: Natural and Technical Sciences], vol. 13, issue 1,129-131 (2008)

[15] Malaschonok G. *Fast Generalized Bruhat Decomposition.* Computer Algebra in Scientific Computing, LNCS 6244, Springer, Berlin 2010. 194-202. distributed memory DOI 10.1007/978-3-642-15274-0_16. arxiv:1702.07242 (2010)

[16] Malaschonok G. *On fast generalized Bruhat decomposition in the domains.* Tambov University Reports. Series: Natural and Technical Sciences. V. 17, Issue 2, P. 544-551. (http://parca.tsutmb.ru/src/MalaschonokGI17_2.pdf) (2012)

[17] Malaschonok G. *Generalized Bruhat decomposition in commutative domains.* Computer Algebra in Scientific Computing. CASC'2013. LNCS 8136, Springer, Heidelberg, 2013, 231-242. DOI 10.1007/978-3-319-02297-0_20. arxiv:1702.07248 (2013)

[18] Malaschonok G., Scherbinin A. *Triangular Decomposition of Matrices in a Domain.* Computer Algebra in Scientific Computing. LNCS 9301, Springer, Switzerland, 2015, 290-304. DOI 10.1007/978-3-319-24021-3_22. arxiv:1702.07243 (2015)

[19] G. Malaschonok and E. Ilchenko, *ecursive Matrix Algorithms in Commutative Domain for Cluster with Distributed Memory,* 2018 Ivannikov Memorial Workshop (IVMEM), Yerevan, Armenia, 2018, pp. 40-46, doi: 10.1109/IVMEM.2018.00015.

[20] Gennadi Malaschonok, Alla Sidko.: Distributed computing: DAP-technology for parallelizing recursive algorithms. Scientific notes of NaUKMA. Computer Science. V.1, pp 25-32. (2018)

[21] Gennadi Malaschonok. Recursive Matrix Algorithms, Distributed Dynamic Control, Scaling, Stability // Proc. of 12th Int. Conf. on Comp. Sci. and Information Technologies (CSIT-2019). September 23-27, 2019, Yerevan.

[22] G.I.Malashonok, A.A. Sidko. Parallel computing on distributed memory: OpenMPI, Java, Math Partner. Textbook. Kyiv: NaUKMA, 2020. - 266 p. ISBN 978-617-7668-14-4

[23] G. Malashonok A. Ivaskevych. Static block-recursive Kholetsky algorithm for a distributed memory cluster. Scientific notes of NaUKMA. Computer Science. T.3. 2020 p. 114-120.

[24] Gennadi Malaschonok. LDU-factorization. E-print 2011.04108, p.1-16. arXiv:2011.04108

Gennadi Malaschonok
National University Kyiv-Mohyla Academy, Kyiv, Ukraine
e-mail: malaschonok@gmail.com