CSIT Conference 2021, Yerevan, Armenia, September 27 - October 1

A supercomputer runtime DAP for matrix block-recursive algorithms

Gennadi Malaschonok
National University of Kyiv-Mohyla Academy
str. Skovorodi, 2 Kiev, Ukraine
email: malaschonok@gmail.com

Alla Sidko
National University of Kyiv-Mohyla Academy
str. Skovorodi, 2 Kiev, Ukraine
email: a.sidko@ukma.edu.ua

***Abstract*—A new runtime for matrix calculations on a supercomputer is proposed. It is designed for recursive matrix algorithms, both dense and sparse matrices, and provides dynamic decentralized control of the computational process. We present a new block-recursive scheme for the Cholesky algorithm as a typical example of a block-recursive algorithm. The results of experiments with different numbers of cores, which demonstrate a high degree of scalability, are described.**

***Keywords*— Supercomputer runtime, dynamic decentralized control for supercomputer, block-recursive matrix algorithm, sparse matrices, runtime DAP, Cholesky algorithm.**

I. Introduction

Modern supercomputer systems with hundreds of thousands of cores have created many new problems in the field of parallel computing (see, for example, [2]). The three main ones are irregular load of equipment, the presence of the growth of the error of numbers during calculations and the possible physical failures of individual processors. Static control of computations in a supercomputer does not allow solving these problems. Therefore, the task of organizing dynamic decentralized control of a distributed computing process in a supercomputer is one of the main tasks today.

A new task insertion extension for PaRSEC was presented in [7], [16]. This scheme of Dynamic Task Discovery (DTD) support shared and distributed memory environments. They compare two programming paradigms: Parameterized Task Graph (PTG) and Dynamic Task Discovery. Where comparable benchmarks exist, the DTD shows better performance compared to the PTG.

Other task-based runtimes (OpenMP [15], StarPU [3], Legion [14], OmpSs [9], OCR [10], HPX [21], SuperGlue [13], QUARK [1], DPLASMA [6]) abstract the available resources to simplify the process of writing a parallel application. A framework for synthesizing communication-efficient distributed-memory parallel programs for block recursive algorithms is presented in [17]. This was one of the earliest systems on the use of block recursive algorithms. In the paper [4], a programming methodology for designing block recursive algorithms on various computer networks was presented. They employ the tensor product notation to formulate computational problems and derive different algorithms on various computer networks.

We continue to develop this area and suggest using another dynamic control scheme for a parallel computing process for block-recursive algorithms. The first approach to creating such parallel programs was a centralized dynamic Low Level Parallelization (LLP) control scheme, in which one of the cluster nodes acted as the dispatcher for the entire computational process [22], [23].

Then a Dynamic Decentralized Parallel (DDP) [12] management scheme was developed. However, the recursion depth was not taken into account in this scheme and it was not possible to switch to a new task until the current task was completed.

The new control scheme is called DAP-scheme (Drop-Amine-Pine) [5]. It differs in that it sequentially expands functions in depth, retaining all states at any nesting level until all calculations in the current computational subtree are completed. This allows any processor to freely switch from one subtask to another, without waiting for the completion of the current subtask.

The second section describes the block-recursive Cholesky decomrposition algorithm.

In the third sections, a description of the DAP runtime is given.

The fourth section presents the results of experiments that demonstrate the scalability of recursive matrix algorithms using the example of matrix multiplication, calculating the inverse matrix, and calculating the Cholesky decomposition.

II. Cholesky decomposition

A symmetric positive-defined matrix A is given. It is required to find a lower triangular matrix L such that equality $A = LL^T$ holds [8].

We propose a block-recursive algorithm version of the Cholesky decomposition algorithm. We divide the matrix A into 4 identical large blocks. Let's denote the blocks of the A matrix $\alpha, \beta, \beta^T, \gamma$, denote the blocks of the L matrix by $a, 0, b, c$ and equate the product LL^T with the matrix A:

$$L \cdot L^T = \begin{pmatrix} a & 0 \\ b & c \end{pmatrix} \cdot \begin{pmatrix} a^T & b^T \\ 0 & c^T \end{pmatrix} =$$

$$\begin{pmatrix} aa^T & ab^T \\ ba^T & bb^T + cc^T \end{pmatrix} = \begin{pmatrix} \alpha & \beta \\ \beta^T & \gamma \end{pmatrix} = A.$$

$$aa^T = \alpha, b = \beta^T (a^{-1})^T, cc^T = \gamma - bb^T$$

To calculate blocks a, b, c, it is necessary to perform block multiplication, transposition, inversion and Cholesky decomposition.

Inverse matrix computation can be avoided if the inversion is computed simultaneously with decomposition. Such a recursive Cholesky procedure should calculate not only the matrix a or b, but also a^{-1} or b^{-1}. In this case, the inverse matrix for matrices L will be such a matrix:

$$L^{-1} = \begin{pmatrix} a^{-1} & 0 \\ -c^{-1}ba^{-1} & c^{-1} \end{pmatrix}.$$

A. Recursive Cholesky algorithm

$$(L, L^{-1}) = Cholesky(A)$$

if$(size(A) == 1\ \&\ A = [\alpha])$ **then return** $([\alpha^{1/2}], [\alpha^{-1/2}])$
else

1) $A- > (\alpha, \beta, \gamma)$ – "we create three blocks"
2) $(a, a_1) = Cholesky(\alpha)$ – "$a_1 = a^{-1}$"
3) $b^T = a_1 * \beta; b = (b^T)^T$
4) $\delta = \gamma - bb^T$
5) $(c, c_1) = Cholesky(\delta)$ – "$c_1 = c^{-1}$"
6) $z = -c_1 * b * a_1$

return $\begin{pmatrix} \begin{pmatrix} a & 0 \\ b & c \end{pmatrix}, \begin{pmatrix} a_1 & 0 \\ z & c_1 \end{pmatrix} \end{pmatrix}$

The graph of this recursive algorithm is shown in the fig. 1.

III. Computation control mechanism

Dichotomous block-recursive algorithms are based on dividing matrices into blocks, to each of which the block-recursive algorithm is again applied.

This happens as long as the blocks remain large enough. When the block size becomes small enough, the sequential algorithms are applied to the blocks.

We call the size of such a small block the "leaf size", it depends on the physical characteristics of the computing device and should be automatically selected for specific equipment.

The connections tree for computational nodes is formed when data is transferred from parent nodes to child nodes. This connections tree can be built according to the graph of recursive algorithm. At first moment all the nodes are free and only root-node takes the whole task and hole list of free nodes.

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.

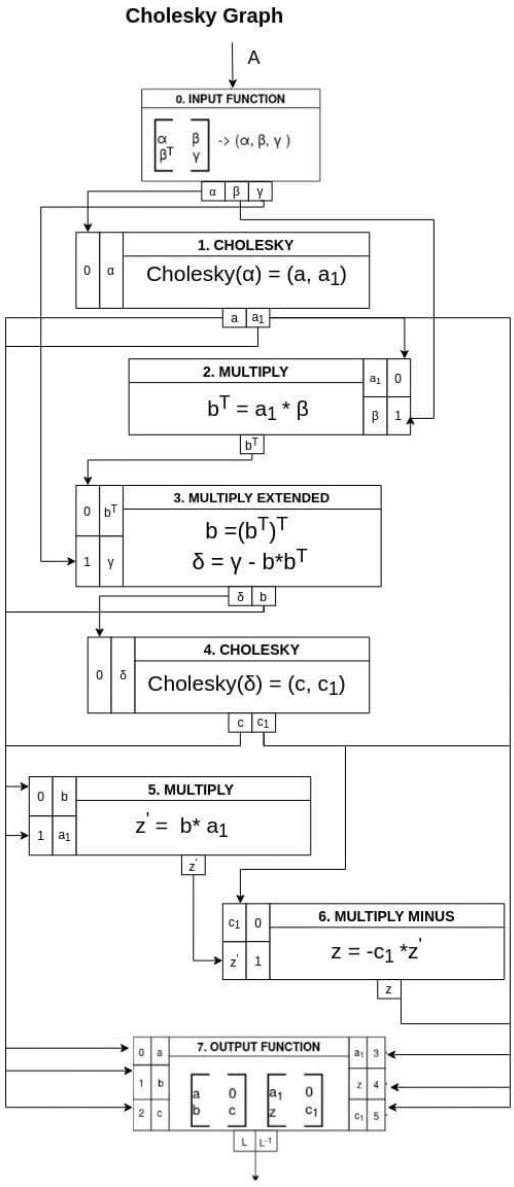

Fig. 1. Graph of Cholesky decomposition

A. The main objects of DAP technology

1) Drop: We divide the computational graph into separate compact subgraphs (drops). In fig. 1 vertices, which are combined into one drop, are rectangles with numbers 1,2, ..,6.

Thus, we define the drops as the smallest components of the computational graph that can be transferred to other processors.

Each Drop has one input function that accepts an array of input data and can perform preliminary simple computations.

Each Drop has one output function that can perform the final simple calculations and form an array of output data.

2) Amine: Before the drop is calculated, we need to expand the corresponding subgraph. This subgraph is called amine. This amine also consists of drops.

According to fig. 1, such an amine contains two drops of multiplication (2, 5), two drops of the Cholesky expansion (1,4), and two special drops with multiplication and some additional operations (3,6).

3) Pine: All amines that are formed in one processor are stored in the general list, which is called Pine.

4) Vokzal: This is the storage location for all drop tasks that await their direction for computation. These tasks are located at different levels. These levels correspond to the depth of recursion for drops.

5) Aerodrome: Each processor that sent a drop task to this processor is called a parent. The list of all parent processors is called Aerodrome.

6) Terminal: The terminal is used to communicate with the child processors. All child processors are stored in the terminal.

B. Two threads

We use two threads: a computational thread and a dispatcher thread. These threads are executed in turn on each core in the cluster. In this case, the main one is the dispatcher thread. It provides all communications with other processors and transfers control to the computational thread, falling asleep for a certain period of time.

C. Protection in case of node failure during the computational process

The parent node submits a job to the child node and should receive the result. If the parent node receives a message about the failure of the child node, then it will redirect this task to another child node. No changes to the computational processes on other nodes are required.

IV. The experiments

All computational experiments were conducted on a cluster MVS-10P CASCADE LAKE of Joint Supercomputer Center of the RAS (2256 cores, based on Intel Xeon Platinum 8268, 28 cores / 48 threads, 2.9GHz, 35.75MB cache, with 96 GB RAM for each processor.)

In table 1, we present the results of three series of experiments that we conducted on a cluster. In each series, the leaf size was kept constant.

TABLE I
SCALABILITY OF CHOLESKY FACTORIZATION FOR BIGDECIMAL WITH 100 DECIMAL DIGITS (TIME IN MIN.)

# cores	1	TLR	8	TLR	64	TLR	512
matrix size	256		512		1024		2048
leaf size 64	1.2	1.14	1.8	2.4	24.8	1.05	28.9
leaf size 128	1.2	1.44	3.6	1.35	8.96	1.31	20.0
leaf size 256	1.1	1.96	8.7	1.21	15.5	1.16	24.0

In the first series, the size of the leaf vertex is 64, in the second – 128, and in the third – 256. In all experiments, we take a constant specific computational load on one computing core. Since the complexity of the algorithm grows according to the cube law, then when the matrix size increased by 2 times, we increased the number of cores by 8 times.

Therefore, we did experiments in which the number of cores was equal to 1, 8, 64 and 512, and the size of the matrix was, respectively, 256, 512, 1024 and 2048. The numbers were taken of type BigDecimal with 100 decimal digits.

We introduce the following concept: transmission loss ratio (TLR). It is a value that shows how the time increased, when the number of cores doubles, and the computational load on one core does not change. As we can see in the best series of experiments (with a leaf size of 128), with an increase in the number of cores by 8 times, the computation time increases by about 2.3 times. More precisely for of these relations we get: $364/121 = 3.0$, $896/364 = 2.4$ and $1996/896 = 2.2$, correspondingly. Therefore, with doubling the size of the cores, the additional costs of data will be equal to $[\sqrt[3]{3.0}, \sqrt[3]{2.4}, \sqrt[3]{2.2}] = [1.44, 1.35, 1.31]$ - this is what we call TLR.

TABLE II
SCALABILITY OF CHOLESKY FACTORIZATION FOR DOUBLE-PRECISION FORMAT (TIME IN MIN.)

# cores	4	TLR	32	TLR	256
matrix size	512		1024		2048
leaf size	32		64		128
density 3%	0.01	1.82	0.06	1.65	0.27
density 30%	0.019	1.65	0.085	1.21	0.15
density 100%	0.018	1.58	0.071	1.22	0.13

TABLE III
SCALABILITY OF MATRIX MULTIPLICATION FOR BIGDECIMAL WITH 100 DECIMAL DIGITS (TIME IN MIN.)

# cores	4	TLR	32	TLR	256
matrix size	512		1024		2048
leaf size	32		64		128
density 3%	0.019	1.85	0.12	1.7	0.59
density 30%	0.26	1.31	0.59	1.56	2.26
density 100%	2.78	1.20	4.8	1.51	16.5

TABLE IV
SCALABILITY OF MATRIX MULTIPLICATION FOR DOUBLE-PRECISION FORMAT (TIME IN MIN.)

# cores	4	TLR	32	TLR	256
matrix size	512		1024		2048
leaf size	32		64		128
density 3%	0.017	1.86	0.11	1.7	0.54
density 30%	0.1	1.59	0.4	1.01	1.41
density 100%	0.73	1.3	1.6	1.4	4.36

Experiments in table 1 demonstrate good computational scalability using DAP-technology on the calculating the factorization using the Cholesky algorithm. The best size of the leaf vertex is 128.

Experiments with Cholesky algorithm, demonstrated in table 2, for the double-precision format.

Experiments, demonstrated in table 3 and table 4, were conducted for algorithm of matrix multiplication for the

TABLE V
SCALABILITY OF MATRIX INVERSION FOR DOUBLE PRECISION FORMAT (TIME IN MIN.)

# cores	4	TLR	32	TLR	256
matrix size	512		1024		2048
leaf size	32		64		128
density 3%	0.006	1.71	0.03	1.54	0.11
density 30%	0.012	1.74	0.063	1.23	0.188
density 100%	0.016	1.6	0.065	1.48	0.21

BigDecimal with 100 digits and for the double-precision format.

Table 5 shows the results of experiments in which the inverse of a triangular matrix was calculated using a block-recursive algorithm and the double-precision format.

V. CONCLUSION

We gave a description of a runtime for dynamic parallelization of recursive algorithms on a distributed memory cluster, described the main objects, and also explained the operation of a two-thread system that runs on each core of the cluster. This runtime is designed to organize computations of any block-recursive algorithms, both with dense and sparse data. It differs in that it sequentially expands functions in depth, retaining all states at any nesting level until all calculations in the current computational subtree are completed. This allows any processor to freely switch from one subtask to another, without waiting for the completion of the current subtask. This runtime makes it possible to organize protection in the event of failure of individual nodes during the computational process.

The scheme was implemented in the Java programming language using the OpenMPI, and its work was tested on the above matrix multiplication, matrix inversion and Cholesky decomposition.

The source code of the program is available at the link: https://bitbucket.org/mathpar/dap/src/master/src/main/java/com/mathpar/parallel/dap/.

ACKNOWLEDGEMENT

We are grateful to the Joint Supercomputer Center of the Russian Academy of Sciences for the opportunity to perform calculations on the MVS-10P supercomputer.

REFERENCES

[1] YarKhan, Asim, *Dynamic Task Execution on Shared and Distributed Memory Architectures.* PhD diss., University of Tennessee, 2012. https://trace.tennessee.edu/utk_graddiss/1575

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

[3] Emmanuel Agullo, Olivier Aumage, Mathieu Faverge, Nathalie Furmento, Florent Pruvost, et al.. Harnessing clusters of hybrid nodes with a sequential task-based programming model. *International Workshop on Parallel Matrix Algorithms and Applications (PMAA 2014)*, Jul 2014, Lugano, Switzerland. vol. 13, pp. 1-14, 2014.

[4] Fan, Min-Hsuan & Huang, Chua-Huang & Chung, Yeh-Ching. A programming methodology for designing block recursive algorithms on various computer networks. *International Conference on Parallel Processing Workshop*, pp. 607-614, 2002. 10.1109/ICPPW.2002.1039783.

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

[6] Massively Parallel Architectures with DPLASMA, *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, pp. 1432-1441, 2011. doi: 10.1109/IPDPS.2011.299.

[7] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault and J. J. Dongarra, PaRSEC: Exploiting Heterogeneity to Enhance Scalability, *Computing in Science & Engineering*, vol. 15, no. 6, pp. 36-45, Nov.-Dec. 2013. doi: 10.1109/MCSE.2013.98.

[8] Gustavson F.G., Karlsson L., Kagstrom B. (2007) Three Algorithms for Cholesky Factorization on Distributed Memory Using Packed Storage. In: Kagstrom B., Elmroth E., Dongarra J., Wasniewski J. (eds) *Applied Parallel Computing. State of the Art in Scientific Computing. PARA 2006.* LNCS, vol 4699. Springer, Berlin, Heidelberg, 2006. doi: 10.1007/978-3-540-75755-9_67

[9] Javier Bueno, Judit Planas, Alejandro Duran, Rosa M. Badia, Xavier Martorell,Eduard Ayguade, and Jesus Labarta. 2012. Productive programming of GPUclusters with OmpSs. *IEEE 26th International Parallel and Distributed Processing Symposium, IPDPS 2012.* pp. 557-568, 2012. doi: 10.1109/IPDPS.2012.58

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

[11] HDFS. Hadoop distributed file system. http://hadoop.apache.org

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

[13] Martin Tillenius. 2015. SuperGlue: A Shared Memory Framework Using Data Versioning for Dependency-Aware Task-Based Parallelization. *SIAM Journal on Scientific Computing*, vol.37, no.6, pp. 617-642, 2015. doi: 10.1137/140989716

[14] M. Bauer, S. Treichler, E. Slaughter and A. Aiken, "Legion: Expressing locality and independence with logical regions,*SC '12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pp. 1-11, 2012. doi: 10.1109/SC.2012.71.

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

[16] 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, pp. 1–8, 2017. doi.org/10.1145/3148226.3148233

[17] 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*, vol. 34, pp.137-153, 1996.

[18] Schonhage A., Strassen V. Schnelle Multiplikation grosser Zahlen. *Computing*, vol. 7, pp. 281-292, 1971. doi:10.1007/BF02242355

[19] Shvachko, Konstantin. Apache Hadoop. The Scalability Update. *Login*, vol. 36, no. 3. pp. 7-13, 2011. ISSN 1044-6397

[20] Strassen V. Gaussian Elimination is not optimal. *Numerische Mathematik*. vol.13, pp. 354-356, 1969.

[21] Heller, T., Kaiser, H. & Iglberger, K. Application of the ParalleX execution model to stencil-based problems. *Comput Sci. Res. and Dev.* vol. 28, pp. 253–261, 2013. doi 10.1007/s00450-012-0217-1

[22] Malashonok G.I., Avetisyan A.I., Valeev Yu.D., Zuev M.S. Parallel algorithms of computer algebra. *Proceedings of the Institute for System Programming,* ed. V.P. Ivannikov. Moscow: ISP RAS, pp. 169-180. 2004.

[23] Malashonok G.I., Valeev Y.D. The control of parallel calculations in recursive symbolic-numerical algorithms. *Parallel computing technologies. International scientific conference (St. Petersburg, January 28 - February 1, 2008)*, Chelyabinsk: Publishing house SUSU, pp. 153-165. 2008. ISBN 978-5-696-03720-2