Mappings and patterns to improve the triangular matrix product on distributed systems

Conference Paper · October 2023

DOI: 10.1109/CLUSTERWorkshops61457.2023.00026

5 authors, including:

María Inmaculada Santamaría Valenzuela
Universidad Politécnica de Madrid
5 PUBLICATIONS **1** CITATION
SEE PROFILE

Rocío Carratalá-Sáez
Universitat Jaume I
21 PUBLICATIONS **59** CITATIONS
SEE PROFILE

Yuri Torres
Universidad de Valladolid
37 PUBLICATIONS **248** CITATIONS
SEE PROFILE

Diego R. Llanos
Universidad de Valladolid
155 PUBLICATIONS **910** CITATIONS
SEE PROFILE

Mappings and patterns to improve the triangular matrix product on distributed systems

Inmaculada Santamaria-Valenzuela, Rocío Carratalá-Sáez, Yuri Torres, Diego R. Llanos, Arturo Gonzalez-Escribano
Dpto. de Ingeniería Informática, Universidad de Valladolid, Valladolid, Spain
{*msantamaria|rocio|yuri.torres|diego|arturo*}*@infor.uva.es*

***Abstract*—Matrix multiplication is one of the most costly linear algebra operations, very often present in scientific computational applications. Current generic linear algebra libraries, such as ScaLAPACK and its recent evolution SLATE, include functionalities for generic and triangular matrix multiplication. They generally rely on block-cyclic partitioning, which has two main advantages. First, it provides good interoperability with other functionalities of the libraries. Second, it provides a good balance of computation and inter-process communications. The focus of these libraries is performance and scalability, targeting even huge number of processes. Nevertheless, many enterprises and computing centers work with commodity clusters or small partitions with a reduced amount of nodes.**

In this paper, we propose and evaluate a combination of data distributions and communication patterns intending to optimize the triangular matrix product in distributed memory systems when targeting commodity clusters (up to approximately 36 nodes). The main four ideas are: Use panels (horizontal or vertical band partitions) instead of tiling; avoid zero-elements in communication buffers; balance the number of elements in communicated buffers; and evaluate the performance when combined with both pipeline and broadcast communication strategies. We compare our implementation performance against the state-of-the-art implementations provided by ScaLAPACK and SLATE. The results show that we outperform both of them. Our proposal is up to 41% faster than ScaLAPACK, and up to 6.7% faster than SLATE.

***Index Terms*—triangular matrices, matrix product, distributed systems, SLATE, ScaLAPACK**

I. Introduction

Classic generic libraries for performing linear algebra operations in distributed memory environments, such as ScaLAPACK [1] and SLATE [2], spread the matrix blocks among the available processors in a cyclic way. This *block-cyclic* matrix distribution leads to efficient fine-grained communications and data alignment for the matrix product computation. This distribution also allows a good interoperability with other functions of the same library. Nevertheless, while this mapping performs very well in the general case, it may be less efficient in specific scenarios, such as the multiplication of triangular matrices.

In this work, we address the matrix-matrix multiplication $C = A \times B$, where A is a lower triangular matrix, offering efficient alternative distribution and communication schemes to those advocated by state-of-the-art libraries. We propose a method that balances communication and computation workloads, thus improving performance for commodity clusters (composed of up to approximately 36 nodes). Our proposal focuses on matrices distribution and communication patterns for lower triangular–full matrix multiplications that can be generalized for upper triangular matrices. We revisit classical mapping methods and communication algorithms, and we propose a combination with data partitioning methods where only non-zero elements are distributed and communicated asynchronously to keep a similar workload in all processes and overlap computation and communication.

To evaluate our proposal, we compare its performance against the distributed memory `dtrmm` operation from ScaLAPACK and SLATE libraries, which perform the triangular-per-full matrix product. Our results show that our solutions outperform the state-of-the-art implementations on commodity clusters.

This work was supported in part by the *Spanish Ministerio de Ciencia e Innovación* and by *the European Regional Development Fund (ERDF)* program of the European Union, under Grant PID2022-142292NB-I00 (NATASHA Project); and *the Junta de Castilla y León - FEDER Grants*, under Grant VA226P20 (PROPHET-2 Project), Junta de Castilla y León, Spain.

II. Our proposal

Let $A \in \mathbb{R}^{m \times n}$ be a triangular matrix, and $B, C \in \mathbb{R}^{m \times n}$ regular matrices. We intend to improve the triangular matrix product $C = A \times B$ by exploring four ideas: Panels usage, avoid sending zeros, consider both pipeline and broadcast communication algorithms, and use partitions with irregular panels to balance the number of non-zero elements in the triangular matrix parts.

It has been proved in the literature that 2D tiling partitioning and communication can scale better for large number of nodes than using panels (horizontal or vertical band partitions) [3]. Our hypothesis is that the use of panels can improve performance in commodity clusters. We propose to distribute A matrix by horizontal panels; and B by vertical panels. Each node computes a whole vertical panel of C. Regarding the communication strategy, at each step of the algorithm, we propose to communicate matrix A parts, and then multiply the received A_i part by the local B_j part. We test two communication strategies: Pipeline (circular shift of A, where A_i sent to processor `(i+1)%num_procs`) and broadcast (at stage `k`, A_k is sent from processor `k` to all processors).

With respect to the management of the communication buffers, we propose two improvements. On the one hand, instead of sending a rectangular full panel (which contains A

zeros), we build a box panel (including the minimal number of zero elements to obtain a rectangular shape) or a trapezoid panel (no zeros); see Figure 1. On the other hand, in order to balance the number of elements sent by each process and the computation at each step, we propose to leverage irregular sized panels that are formed by approximately the same number of non-zero elements; see Figure 2.

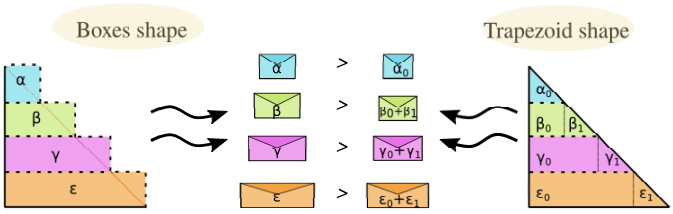

Fig. 1. Sample of the boxes and trapezoid shapes built to communicate the triangular matrix data.

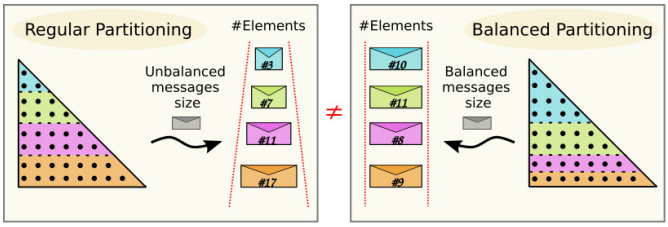

Fig. 2. Sample of the regular and balanced partitioning of the triangular matrix data.

III. Performance evaluation

All the tests have been conducted in the SCAYLE supercomputer[1], in particular in its Cascade Lake server, which is equipped with 38 servers with 192 GB RAM and Infiniband HDR 100 Gbps interconnection Network. Each node is composed of 2 Intel Xeon Gold 6240 processors 2.6 GHz. All the implementations have been compiled using GCC 9.4.0.

To evaluate the performance of our proposal and to position it with respect to the state-of-the-art libraries, we compare the execution times of our proposal with both ScaLAPACK and SLATE. We use ScaLAPACK from Intel's oneAPI 2021.3.0, and SLATE release 2023.06.00. We measure the elapsed time to compute the triangular-full matrix product with square matrices of dimension 30,000×30,000. As we base our proposal in four main ideas, we test eight different versions of our implementation: Choosing to use regular or balanced buffers, pipeline or broadcast strategy, and boxes or trapezoid shapes. To do the local computation at each step, we use the `gemm` and `trmm` routines from Intel's oneAPI 2021.3.0 MKL. Figure 3 shows the execution time (in seconds) obtained by ScaLAPACK, SLATE and the eight versions of our proposal.

	Execution time (seconds)			
Number of MPI processes	**9**	**16**	**25**	**36**
SLATE	**145,25**	**83,09**	**54,43**	**39,90**
ScaLAPACK	**225,67**	**131,54**	**87,26**	**64,50**
regular, pipeline, boxes	**137,06**	**79,21**	**53,11**	**39,20**
regular, pipeline, trapezoid	**137,27**	**79,33**	**53,02**	**39,17**
regular, bcast, boxes	**135,96**	**77,47**	**52,70**	**38,70**
regular, bcast, trapezoid	**135,73**	**77,53**	**52,88**	**38,60**
balanced, pipeline, boxes	**137,81**	**79,75**	**53,71**	**39,57**
balanced, pipeline, trapezoid	**137,61**	**79,62**	**53,48**	**39,40**
balanced, bcast, boxes	**135,87**	**78,40**	**52,55**	**38,30**
balanced, bcast, trapezoid	**136,05**	**78,19**	**52,54**	**38,16**

Fig. 3. Elapsed time (seconds) to compute the triangular matrix product $C = A \times B$ with A being a lower triangular matrix, and B a regular matrix, both of them of dimension $30,000 \times 30,000$.

IV. Conclusion

In this work, we present an alternative combination of data partitions, mappings, and communication schemes to compute triangular-full matrix products in distributed commodity clusters. We show that this combination can outperform the state-of-the-art implementations in linear algebra libraries, such ScaLAPACK and SLATE, for clusters up to 36 nodes. In particular, in the case of ScaLAPACK, in the best case we attain a reduction of up to 41% of the elapsed time to compute the triangular-full matrix product of square matrices of dimension 30,000. Regarding SLATE, although smaller, we still observe performance benefits using our proposal up to 6.7%. With respect to the different variants of our proposal, in general, broadcasting the local matrices with trapezoid balanced shapes is the best option. Although, the difference between boxes and trapezoid shapes is small, the reduced number of elements communicated when using trapezoids slightly pays off the extra cost of marshalling with more complex trapezoid buffers.

Although tiling based partitionings scale better on the long-term than those based on panels, we have shown that using specific partitioning, mapping, and communicating techniques for panels can offer performance benefits for the small number of nodes that can be found in commodity clusters, or small partitions of bigger computing center installations.

Currently we leverage CBLACS kernels to compute the gemm and trmm operations. SLATE, for example, uses more efficient multithreaded versions of these kernels. As part of the future work, we plan to explore the same comparisons presented in this work using comparable multithreaded linear algebra kernels. We also plan to use this partition schemes directly in SLATE.

[1] https://www.scayle.es/

References

[1] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA: Society for Industrial and Applied Mathematics, 1997.

[2] M. Gates, J. Kurzak, A. Charara, A. YarKhan, and J. Dongarra, "Slate: Design of a modern distributed and accelerated linear algebra library," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[3] L. S. Blackford, J. Choi, A. Cleary, A. Petitet, R. C. Whaley, J. Demmel, I. Dhillon, K. Stanley, J. Dongarra, S. Hammarling, G. Henry, and D. Walker, "ScaLAPACK: A portable linear algebra library for distributed memory computers - design issues and performance," in *Proceedings of the 1996 ACM/IEEE Conference on Supercomputing*, Supercomputing '96, (USA), p. 5–es, IEEE Computer Society, 1996.