ParCFD'2020
32^{nd} International Conference on Parallel Computational Fluid Dynamics
May-11-13 2020, Nice FRANCE

CHALLENGES IN RESHAPING LINEAR ALGEBRA LIBRARIES FOR HPC/AI WORKLOADS

Hatem LTAIEF and David KEYES

Extreme Computing Research Center, KAUST, 23955 Thuwal, Saudi Arabia
e-mail: first.last@kaust.edu.sa, web page: https://cemse.kaust.edu.sa/ecrc

Key words: Numerical Linear Algebra, Artificial Intelligence, High Performance Computing, Massively Parallel Algorithms, Performance Optimizations.

Abstract. Traditional HPC simulations and AI / Big Data applications face similar challenges when solving extreme-scale scientific problems: bulk synchronous parallelism, expensive data motion, high algorithmic complexity and large memory footprint. Processors and memory technology scaling have mitigated these challenges thanks to an exponential growth in processor performance but only a constant increase in memory speed and capacity. The free lunch is perhaps over as we approach the hard physical limit of silicon. The energy efficiency gap between communication and computation keeps widening and has even forced the hardware and software communities for an immediate action of co-design. We describe the challenges encountered during the last 15-year journey of reshaping high performance linear algebra libraries for massively parallel systems. We explore disruptive numerical algorithms and programming models required to continue supporting HPC applications as well as emerging AI workloads at the dawn of the exascale age.

1 A RENAISSANCE IN COMPUTATIONAL LINEAR ALGEBRA

A renaissance has come to computational linear algebra in form of hierarchically low rank matrices (henceforth "$\mathcal{H}$-matrices"). They are useful in a wide variety of applications leading to dense matrices, such as mechanics and electrostatics formulated in terms of Green's functions, maximum likelihood in spatial statistics built on covariance matrices, optimization based on Hessians, and even applications of sparse matrices during which dense Schur complements are formed. Formally dense operators are often "data sparse" in the sense that their input-to-output maps can be mediated to high accuracy in far less than n^2 operations, and inverted in far less than $O(n^3)$ operations. Indeed, the curse of dimensionality can be mitigated in these applications and others by the blessing of low rank. The emergence of algorithms exploiting hierarchical low rank over the past two decades, since the seminal work of Hackbusch [5] and Tyrtyshnikov [8], could hardly come at a more auspicious time in terms of computer architecture. A main motivation is the decreasing ratio of memory bandwidth to processing power [7] and the growing latency in clock cycles of accessing an element of deep memory, which can be a thousand or more for DRAM. Data sparsity implies that relatively small cache memories can hold relatively highly accurate representations of operators. The savings in latency from residing high on

the memory hierarchy is even more important than the savings in operation count from working directly with the compressed representation.

2 TILE LOW-RANK MATRIX COMPUTATIONS

Tile low rank linear algebra was inspired by the block low rank (BLR) compression of Schur complements of elliptic PDE operators in [3]. A $m \times n$ matrix A is practically of low rank if $A = UV^T + E$, where U is $m \times k$, V is $n \times k$, for $k < mn/(m+n)$, where $||E||_2 < \epsilon$ is small enough to be neglected and rank k depends upon the accuracy tolerance ϵ.

In a typical TLR matrix application, A is partitioned in advance into blocks of uniform size related to the level of the memory hierarchy in which they should fit and/or the number of threads available on the node, with due consideration of ordering to cluster degrees of freedom with the strongest coupling along the primary and possibly other diagonals. Each tile that is believed to be a candidate for low rank representation is then independently compressed using any of a variety of algorithms to determine an acceptable k, typically by considerations local to a tile. Ranks may vary across tiles; hence the task of compression may not be load-balanced across tiles, nor may be the subsequent tasks of manipulating tiles within the context of a standard tile algorithm. The latter may require matrix-matrix multiplications, matrix-matrix additions, or the application of the inverse of a full-rank tile to other tiles. In the context of tile algorithms, this is not a major drawback because they are typically executed via a task-based dynamic runtime system based on a directed acyclic graph (DAG).

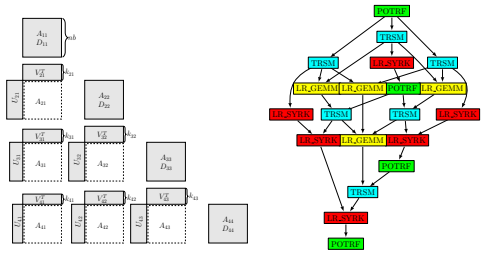

Figure 1: An originally dense symmetric positive definite matrix decomposed into tiles (left) and a DAG for its Cholesky factorization (right).

For a dense symmetric positive definite matrix, a tile-based Cholesky factorization defines a sequence of diagonal block factorizations, column block scalings by diagonal block inverses, and block row multiplication and addition updates. A sample DAG for the 4×4 blocked symmetric matrix A on the left in Figure 1 is shown to the right. Colored rectangular nodes represent tasks and the arrows data dependencies. Each of the four diagonal block factorizations (POTRF, in green) is followed by block updates to its own lower subtriangle, through the last block A_{44}, which is its own subtriangle.

A TLR data structure begins with the tile decomposition of A. Its diagonal blocks, D_{ii}, are the same as those of A. We then replace the off-diagonal blocks, A_{ij} for $i > j$, with low rank factorizations, $U_{ij}V_{ij}^T$, where the factors have rank $k_{ij} < n_b$, where n_b is the (traditionally uniform) block size. Preferred compression routines are the Randomized SVD [6] or adaptive cross-approximation (ACA) [4]. Whenever a low rank tile is updated, it requires recompression.

3 PERFORMANCE RESULTS

TLR technique has provided another one to two orders of magnitude of performance in the last two years, depending upon compressibility. TLR is the main compressed data structure implemented in the Hierarchical Computations on Manycore Architectures (HiCMA[1]) library [1], which is described herein.

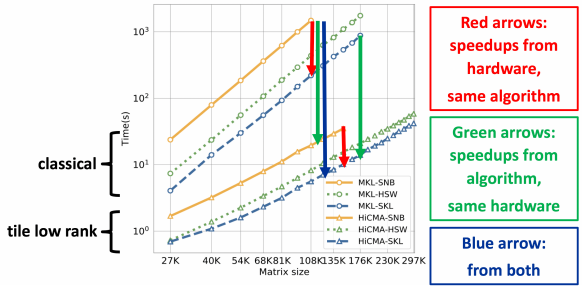

Figure 2: Shared-memory implementations of DPOTRF on three generations of Intel hardware and two generations of algorithms.

Figure 2 illustrates this performance boost for the DPOTRF kernel on a two-dimensional geospatial covariance matrix represented to tolerance 10^{-8} or better in Froebenius norm for each tile for three generations of Intel manycore and two generations of algorithms, for a range matrix sizes from 27K up to 297K, as memory capacity allows. The classical algorithm follows an $O(n^3)$ scaling and can be extended only up to matrices of dimension 108K on an Intel Sandy Bridge processor using Intel's Matrix Kernel Library (MKL). Successive generations of Intel processor hardware, namely Haswell and Skylake, provide runtime improvements (red arrows) and memory capacity improvements while following the same scaling. The TLR algorithm as implemented in the HiCMA Library [1] shows closer to quadratic scaling in runtime with problem size, with significantly greater problem-size accommodation and runtime reductions (green arrows) on the same hardware. The blue arrow shows the product of the hardware and algorithmic advances, already more than two orders of magnitude for matrices of dimension 108K and growing with size. We focus on factorization time only since the time to generate and compress relative to the factorization time decreases due to the difference in the asymptotic complexities of the respective phases [2].

Still larger matrix sizes can be accommodated by distributed-memory versions of (P)DPOTRF and HiCMA, using MPI, as shown on the left in Figure 3. With the memory savings of TLR, larger problems can be accommodated for a given node count, and nearly two orders of magnitude of runtime improvement come from algorithmic improvement, better than from the comparable ratios of concurrency. A quantification of the memory footprint improvement of TLR is shown on the right in Figure 3. A symmetric matrix of dimension 1M

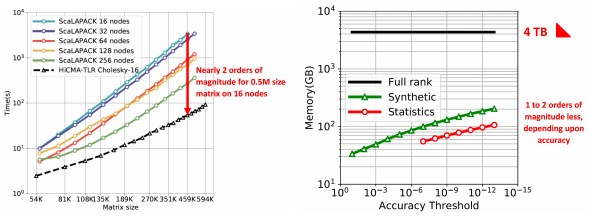

Figure 3: Distributed-memory implementations of DPOTRF on a Haswell-based Cray XC40, on 16 through 256 nodes for ScaLAPACK and TLR on 16 nodes (left). Memory footprints of double precision synthetic and covariance matrices of dimension 1M for a range of block tolerance thresholds, compared with fully dense (right).

[1]HiCMA is a transliteration of the Arabic word for "wisdom"

stored in the lower triangle in 8-Byte precision requires 4 TB. With a tight tolerance of 10^{-13}, worthy of essentially full precision, TLR enables more than an order of magnitude of storage savings. Depending upon the compressibility of the matrix and the accuracy threshold, nearly two orders of magnitude of storage savings are possible. The matrix is generated tile-by-tile using a user-defined matrix kernel and compressed on the fly. Therefore, at no single point in time does the full dense matrix need to reside in main memory.

4 CONCLUSIONS

We have demonstrated the attractiveness of TLR because it can be retrofit into existing tile-based shared-memory and distributed-memory software by simply overloading the fully dense matrix kernel operations with their low rank counterparts, while providing decent performance improvements.

REFERENCES

[1] Abdulah, S. and Akbudak, K. and Boukaram, W. and Charara, A. and Keyes, D. and Ltaief, H. and Mikhalev, A. and Sukkari, D. and Turkiyyah, G. Hierarchical Computations on Manycore Architectures (HiCMA), 2019. Available at http://github.com/ecrc/hicma.

[2] K. Akbudak, H. Ltaief, A. Mikhalev, A. Charara, A. Esposito, and D. Keyes. Exploiting data sparsity for large-scale matrix computations. In M. Aldinucci, L. Padovani, and M. Torquati, editors, *Euro-Par 2018: Parallel Processing*, pages 721–734, Cham, 2018. Springer International Publishing.

[3] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. Improving multifrontal methods by means of block low-rank representations. *SIAM Journal on Scientific Computing*, 37(3):A1451–A1474, 2015.

[4] M. Bebendorf and S. Rjasanow. Adaptive low-rank approximation of collocation matrices. *Computing*, 70:1–24, 2003.

[5] W. Hackbusch. A sparse matrix arithmetic based on H-matrices, part I: Introduction to H-matrices. *Computing*, 62:89–108, 1999.

[6] N. Halko, P. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[7] S. Rumley, N. Bahadori, R. Polster, S. Hammond, D. Calhoun, K. Wen, A. Rodrigues, and K. Bergman. Optical interconnects for extreme scale computing systems. *Parallel Computing*, 53:367–380, 2017.

[8] E. Tyrtyshnikov. Incomplete cross approximation in the mosaic-skeleton method. *Computing*, 42:367–380, 2000.