Exploiting Data Sparsity for Large-Scale Matrix Computations

Kadir Akbudak[1], Hatem Ltaief[1], Aleksandr Mikhalev[1(✉)], Ali Charara[1], Aniello Esposito[2], and David Keyes[1]

[1] Extreme Computing Research Center, Division of Computer, Electrical, and Mathematical Sciences and Engineering, King Abdullah University of Science and Technology, Thuwal Jeddah 23955, Kingdom of Saudi Arabia
{kadir.akbudak,hatem.ltaief, aleksandr.mikhalev,ali.charara,david.keyes}@kaust.edu.sa

[2] Cray EMEA Research Lab, Bristol, UK
esposito@cray.com

Abstract. Exploiting data sparsity in dense matrices is an algorithmic bridge between architectures that are increasingly memory-austere on a per-core basis and extreme-scale applications. In this work, we leverage the Hierarchical matrix Computations on Manycore Architectures (HiCMA) library in order to tackle this challenging problem by achieving significant reductions in time to solution and memory footprint, while preserving a specified accuracy requirement of the application. We have extended HiCMA to provide a high-performance implementation on distributed-memory systems of one of the most widely used matrix factorization in large-scale scientific applications, i.e., the Cholesky factorization. It employs the tile low-rank data format to compress the dense data-sparse off-diagonal tiles of the matrix. It then decomposes the matrix computations into interdependent tasks and relies on the dynamic runtime system StarPU for asynchronous out-of-order scheduling, while allowing high user productivity. Performance comparisons and memory footprint on matrix dimensions up to eleven million show a performance gain and memory saving of more than an order of magnitude for both metrics on thousands of cores, against state-of-the-art open-source and vendor optimized numerical libraries. This represents an important milestone in enabling large-scale matrix computations toward solving big data problems in geospatial statistics for climate/weather forecasting applications.

1 Introduction

State-of-the-art dense linear algebra libraries are confronting memory capacity limits and/or are not able to produce solutions in reasonable times, when performing dense computations (e.g., matrix factorizations and solutions) on large

M. Aldinucci et al. (Eds.): Euro-Par 2018, LNCS 11014, pp. 721–734, 2018.
https://doi.org/10.1007/978-3-319-96983-1_51

matrix of size n, with n in the billions. The current trend of hardware over-provisioning in terms of floating-point units (e.g., with wide SIMD implementations) and the increase of memory capacity (e.g., with new fast non-volatile memory layer) are not sufficient to cope with the emergence of big data problems involving dense matrices due to the prohibitive cubic algorithmic complexity $O(n^3)$ and the expensive quadratic memory footprint $O(n^2)$. To overcome both challenges, matrix approximations may be considered as an effective remedy, as long as the numerical fidelity of the original problem is preserved.

This paper introduces the Hierarchical matrix Computations on Manycore Architectures (`HiCMA`) library, which exploits the data sparsity structure of dense matrices on shared and shared to distributed-memory systems. In particular, the class of covariance-based matrices emerges from various scientific big data applications in environmental applications, including geospatial statistics for climate/weather forecasting [29,30]. Under such an apparently dense matrix representation lies a family of sparse representations. `HiCMA` currently employs a tile low-rank (TLR) data format, which leverages the data descriptor behind the traditional dense/plain tile format [1,2,13]. The idea consists in compressing the off-diagonal tiles, and retaining the most significant singular values with their associated singular vectors up to an application-dependent accuracy threshold. `HiCMA` can then perform matrix operations on these compressed tiles. A dynamic runtime system `StarPU` [10] orchestrates the various computational tasks representing the nodes of a directed acyclic graph, and asynchronously schedules them on the available processing units in an out-of-order fashion, while carefully tracking their data dependencies. This systematic approach enhances the productivity for the library development, while facilitating the code deployment from shared to distributed-memory systems [4].

We assess the numerical robustness and parallel performance of `HiCMA` using two matrix kernels. The first one is a synthetic matrix kernel, and has been inspired from wave-based frequency domain matrix equation problems. It gives a useful flexibility, since it permits to generate customized matrices with various rank sizes and accuracy thresholds. This flexibility can be employed to avoid stressing the solver infrastructure. The second kernel corresponds to a realistic application coming from the family of parametrizable Matérn covariance function [22], and represents the state-of-the-art in modeling geostatistics and spatial statistics [16]. The resulting covariance matrices for both aforementioned kernels are symmetric and positive-definite. The Cholesky factorization is the core operation when solving linear systems of equations for the former or calculating the matrix determinant for the latter. The Cholesky factorization reduces a symmetric positive-definite matrix into lower or upper triangular form and is usually used as a pre-processing step toward solving dense linear system of equations. Thanks to the resulting low arithmetic intensity of the numerical kernels, `HiCMA` is able to translate the original dense compute-bound application into a data-sparse communication-bound on distributed-memory systems. While time to solution is significantly reduced, the bottlenecks are shifted and data traffic reduction may rapidly become central in strong scaling mode of operation, as usually observed for sparse computations.

We report performance comparisons and memory footprint on matrix dimensions up to eleven million and 16,000 cores. We show a gain of more than an order of magnitude for both metrics against state-of-the-art open-source and vendor optimized numerical libraries, when applicable. In these experiments, we employ a threshold which preserves the specific accuracy requirement of the application while removing the irrelevant information data from the matrix. We also provide a comprehensive profiling and tracing results to identify current performance bottlenecks in `HiCMA`. Last but not least, we show preliminary power profiling results to study the impact of the numerical accuracy on the overall energy consumption. The energy consumption stands as a new critical metric to monitor and optimize, especially when solving big data problems in geospatial statistics for climate/weather forecasting applications.

The remainder of the paper is organized as follows. Section 2 provides a bibliography in hierarchical low-rank matrix computations and our research contributions. Section 3 introduces and describes the `HiCMA` software infrastructure for solving large-scale data-sparse problems. Section 4 defines the kernels for synthetic matrix generations and real world applications from climate/weather forecasting applications based on a geospatial statistics approach. Section 5 gives implementation details of the tile low-rank Cholesky, which relies on the `StarPU` dynamic runtime system. Section 6 presents the results and a comprehensive performance analysis. It compares our implementation against existing state-of-the-art implementations on distributed-memory system. We conclude in Sect. 7.

2 Related Work

Discovered around two decades ago [17–19,21,31], hierarchical low-rank matrix approximations are currently a leading algorithmic trend in the scientific community to solve large-scale data-sparse problems. Based on recursive formulations, they exploit the data sparsity of the matrix by compressing the low-rank off-diagonal blocks using an adequate data storage format such as HODLR [6,9], $\mathcal{H}$ [20], HSS [5,27] and $\mathcal{H}^2$ [12]. The aforementioned data compression formats are characterized by linear and log linear upper bounds for their algorithmic complexities. The resulting low arithmetic intensity of the kernel in addition to the recursive formulation impede their parallel performance. They turn out to be difficult to implement, and not amenable to effectively map on manycore shared and distributed-memory systems, due to their fork-join paradigm.

More recently, with the emergence of asynchronous task-based programming models, these hierarchical low-rank matrix approximations algorithms have been revisited by flattening their recursions and exposing them to task-based runtime systems such as Intel Threading Building Blocks (Intel TBB) [24] and OpenMP [4]. While these dynamic runtimes permit to mitigate the overhead from the bus bandwidth saturation on single shared-memory nodes, they do not support distributed-memory systems. Moreover, the authors in [14] have also demonstrated the importance of flattening the recursion during the compression of $\mathcal{H}^2$-matrices, when targeting massively parallel GPU accelerators. Since

memory is a scarce resource on the GPU, porting the $\mathcal{H}^2$-matrix approximation kernel to multiple GPUs appears mandatory but seems to be a very tedious exercise, due to the complex handmade memory management across GPUs.

Another data compression format has been introduced [7], i.e., the block low-rank data format, which is a subset of the $\mathcal{H}$-matrix class of data-sparse approximation. It consists of splitting the dense matrix into blocks and to perform low-rank matrix computations, while compressing the final results on-the-fly. Although distributed-memory systems do not appear as a hostile environment anymore with this new format in the context of sparse direct solvers [8], there may be still two main limitations: the lack of a systematic approach to schedule the computational tasks onto resources and the high memory footprint, since the matrix is not compressed initially, but rather gets compressed as the computation goes.

This paper introduces the `HiCMA` library, the first implementation of task-based tile low-rank Cholesky factorization on distributed-memory systems. Compared to the initial implementation on shared-memory environment [4] based on `OpenMP`, this paper uses instead the `StarPU` [10] dynamic runtime system to asynchronously schedule computational tasks across interconnected remote nodes. This highly productive association of task-based programming model with dynamic runtime systems permits to tackle in a systematic way advanced hardware systems by abstracting their complexity from the numerical library developers. This separation of concerns between hardware and software facilitates in solving large-scale simulations and allows porting `HiCMA` onto large resources and large problems sizes, i.e., 16,000 cores and 11 million, respectively. We have also conducted performance and power profiling analysis to provide further insights when scheduling this new class of algorithms for hierarchical low-rank matrix computations. The `HiCMA` software library[1] has been released and is freely available for public download under the open-source modified BSD license.

3 The `HiCMA` Software Library

The `HiCMA` software library provides a high-performance implementation of the Cholesky factorization for symmetric positive-definite matrices with a data sparse structure. A complete list of `HiCMA` features can be found at https://github.com/ecrc/hicma. `HiCMA` is rooted in tile algorithms for dense linear algebra [2], which split the matrix into dense tiles. `HiCMA` leverages the tile data descriptor in order to support the new tile low-rank (TLR) compression format. While this data descriptor is paramount to expose parallelism, it is also critical for the data management in distributed-memory environment [1,13]. `HiCMA` adopts a flattened algorithmic design to bring to the fore the task parallelism, as opposed to plain recursive approach, which has constituted the basis for performance of previous $\mathcal{H}$-matrix libraries [18,19,21].

[1] https://github.com/ecrc/hicma.

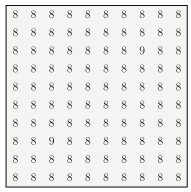
Once the matrix has been divided into tiles, `HiCMA` relies on the `STARS-H` library [2], a high performance $\mathcal{H}$-matrix market, to generate and compress each tile independently. This allows to create the TLR matrix in compressed format, without having a global dense representation, and therefore, opens opportunities to solve large-scale applications, thanks to a low memory footprint. This may eventually become cumbersome even for sparse solvers [7], when dealing with high dimensional problems, since the global intermediate dense matrices are explicitly generated. Figure 3(b) in [4] sketches the TLR matrix after compressing on-the-fly each tile with a specific application-dependent fixed accuracy. This may result into low-rank tiles with non-uniform ranks to maintain the overall expected accuracy. Although the scope of `HiCMA` described in this paper focuses on dense covariance-based scientific applications, it may have a broader impact. Indeed, as previously mentioned, it can also service sparse direct solvers, i.e., supernodal [23,28] and multifrontal numerical methods [8,26], during the low-rank Schur complement calculations on the fronts/supernodes, which are the crux of sparse computations. Furthermore, the fixed rank feature of `HiCMA`, as shown in Fig. 3(a) in [4], allows to generate TLR matrices with uniform ranks across all low-rank tiles. This rough approximations may be of high interest for speeding up sparse preconditioners (e.g., incomplete Cholesky/LU factorizations) during iterative solvers, since important *a priori* assumptions can be made to optimize and improve parallel performance.

4 Definition of Matrix Kernels

The matrix kernel is a function that generates matrix entries, i.e., $A_{ij} = f(x_i, y_j)$, from two sets $\{x_i\}$ and $\{y_j\}$. Typically, the matrix kernel function $f(x, y)$ calculates the interaction between two objects x and y, using a distance-based formulation. Although some matrix kernels may lead to sparse matrices, we investigate matrix kernels that translate into dense matrices.

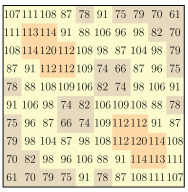
Synthetic Matrix Kernel. The first matrix kernel is a synthetic one, inspired from the core matrix kernel of wave-based matrix equations, as in electrodynamics, electromagnetic and acoustic applications. The matrix kernel can be defined as $\mathrm{f}(x, y) = \frac{\sin(\lambda r(x,y))}{r(x,y)}$, where λ is a wave number and $r(x, y)$ is an Euclidian distance between x and y. In fact, this corresponds to the imaginary part of a fundamental solution $\frac{e^{i\lambda r}}{r}$ of the Helmholtz equation. This modified function is very convenient, since the wave number has a direct impact of the rank distribution on the TLR matrix. This permits to test the numerical robustness of `HiCMA` with a large number of rank configurations. Figures 1(a)-(d) depict the rank distribution for various wave numbers λ on a matrix of size 2500×2500 with an accuracy threshold set to 10^{-9}. It shows a homogeneity among rank sizes of the off-diagonal tiles for a given λ, while it displays rank growth as λ increases.

[2] https://github.com/ecrc/stars-h.

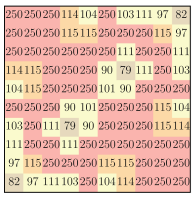


(a) Synthetic λ=1.

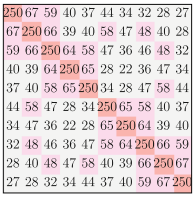
(b) Synthetic λ=10.



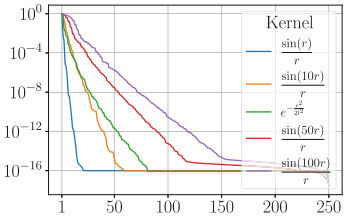
(c) Synthetic λ=50.



(d) Synthetic λ=100.



(e) Real-world.



(f) Singular value distributions.

Fig. 1. Rank distributions for the synthetic matrix kernel with different wave numbers λ and acc=10^{-9} (a)-(d). Rank distributions the spatial statistics applications with acc=10^{-8} (e). Distribution of normalized singular values for both matrix kernels on the bottom-left off-diagonal tile (f). Matrix size $N = 2500$.

Matérn Matrix Kernel for Covariance Problems. The Matérn matrix kernel is at the core of spatial statistics and is used as the state-of-the-art model to drive climate/weather forecasting applications [22]. We have implemented the square exponential variant of this Matérn matrix kernel to demonstrate the effectiveness of our TLR approach for solving real world applications. The square exponential kernel can be defined as $f(r) = e^{-\frac{r^2}{2l^2}}$, where r is a distance between spatial points and l is a correlation length. For the experiments presented in the paper, we set the correlation length to 0.1. The resulting TLR covariance matrix is then used to evaluate the maximum likelihood function during an optimization and iterative procedure, as explained in [4]. Each iteration requires the calculation of the matrix determinant involving the Cholesky factorization, which is the most time-consuming phase. Figure 1(e) shows the rank distributions on a matrix of size 2500×2500 with tile size 250 for the square exponential kernel. The ranks are not too disparate for the corresponding accuracy of 10^{-8}.

The singular value distributions for the tile located at the bottom-left of the TLR matrix generated from each type of matrix kernels is depicted in Fig. 1(f). Their distributions highlight an exponential decay, and therefore, reveal the data sparsity structure of such matrices. It is also clear that, given these rank heatmaps, data compression formats with weak admissibility conditions (i.e., HODLR and HSS) may not be appropriate for this application, due to nested dissection, which operates only for diagonal super tiles. The off-diagonal super

tiles may then engender excessive larger ranks after compression, which may eventually have a negative impact on performance and memory footprint.

5 Implementation Details

This section provides insights into two main ingredients for porting the `HiCMA` library to distributed-memory systems: the data descriptor and the `StarPU` dynamic runtime system.

The Data Descriptor. The data descriptor defines the backbone of `HiCMA`, as it dictates how the data management takes place across the distributed-memory computational nodes. Originally developed for `ScaLAPACK` [11] and inspired later `DPLASMA`'s [13], the descriptor draws how the data is scattered among processing units following the classical two-dimensional block cyclic distribution to ensure load balancing between nodes. `HiCMA` leverages this single descriptor for dense matrices and creates three descriptors to carry on computations over the compressed bases of each tile, calculated by using the randomized SVD compression algorithm from the `STARS-H` library. These bases, i.e., U and V, are of sizes nb-by-k and k-by-nb, respectively, with nb the tile size and k the tile rank. The first descriptor stitches the rectangular bases U and V^T of each tile together. Since k is not known *a priori*, we define a maximum rank ($maxrk$), which can be tuned for memory footprint as well as performance. The second descriptor contains the actual ranks of each tile after compression and gets updated during the computation accordingly. The last descriptor store information about the dense diagonal tiles. The main challenge with these descriptors is that they enforce each data structure they inherently describe to be homogeneous across all tiles. While the rank and dense diagonal tiles descriptors are obviously important to maintain for numerical correctness, the descriptor for the off-diagonal tiles has a direct impact on the overall communication volume. Therefore, tuning the $maxrk$ parameter is mandatory. However, dense data-sparse matrices with a large disparity in the ranks of the off-diagonal tiles may encounter performance bottlenecks, due to excessive remote data transfers. One possible remedy is to implement a fine-grained descriptor for each single tile, as explained in [25].

The `StarPU` Dynamic Runtime System. The `StarPU` dynamic runtime system [10] maps a sequential task-based application onto a complex underlying parallel shared and/or distributed-memory systems. This allows endusers to focus on getting correctness from their sequential implementations and leave the challenging part of porting their codes to parallel environment to the runtime. The pseudo-code for the task-based TLR Cholesky factorizations is presented in Algorithm 1. We refer readers to [4] for the full description of the sequential kernels. The Insert_Task API encapsulates the task, its parameters with their data directions, i.e., read and/or write (STARPU_R, STARPU_W and STARPU_RW). Not only does `StarPU` execute its tasks asynchronously within

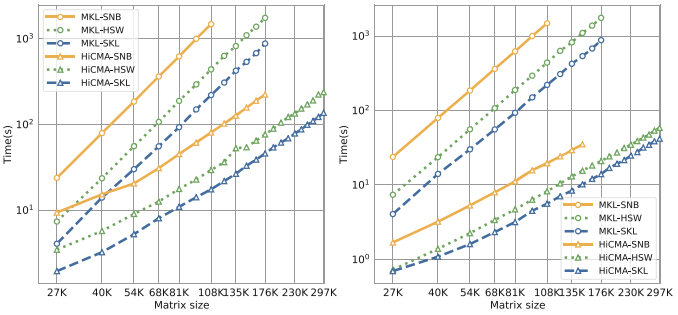
Algorithm 1. `hicma_dpotrf` (HicmaLower, D, U, V, N, nb, rank, acc)

```
p = N / nb
for k = 1 to p do
    StarPU_Insert_Task(hcore_dpotrf, HicmaLower, STARPU_R, D(k), rank, acc)
    for i = k+1 to p do
        StarPU_Insert_Task(hcore_dtrsm, STARPU_RW, V(i,k), STARPU_R, D(k,k))
    end for
    for j = k+1 to p do
        StarPU_Insert_Task(hcore_dsyrk, STARPU_RW, D(j), STARPU_R, U(j,k), STARPU_R, V(j,k))
        for i = j+1 to p do
            StarPU_Insert_Task(hcore_dgemm, STARPU_R, U(i,k), V(i,k), STARPU_R, U(j,k), STARPU_R, V(j,k),
            STARPU_RW, U(i,j), STARPU_RW, V(i,j), rank, acc)
        end for
    end for
end for
```

each node in an out-of-order fashion, but it also performs remote non-blocking point-to-point MPI communications to mitigate the data movement overhead by overlapping it with computations.

6 Performance Results

Our experiments have been conducted on two Cray systems. The first one, codenamed *Shaheen-2*, is a Cray XC40 system with the Cray Aries network interconnect, which implements a Dragonfly network topology. It has 6174 compute nodes, each with two-socket 16-core Intel Haswell running at 2.30 GHz and 128 GB of DDR3 main memory. The second system, codenamed *Cray-SKL*, has roughly 300 nodes with mixed stock keeping units (SKUs) The majority of nodes has at least two-socket 20-core Intel Skylake and at least 192GB of DDR4 memory, where the base frequency of the different SKUs varies between 2.1 GHz and 2.4 GHz. `HiCMA` and `StarPU` have been compiled with Intel compiler suites v16.3.3.210 and v17.0.4.196 on *Shaheen-2* and *Cray-SKL*, respectively. Calculations have been performed in double precision arithmetic and the best performance after three runs is reported.



(a) Synthetic ($\lambda = 100$). (b) Statistics.

Fig. 2. Time to solution of Intel MKL's dense `dpotrf` and `hicma_dpotrf` for both matrix kernels on Sandy Bridge, Haswell, and Skylake shared-memory systems.

Figure 2 shows the performance of the dense and TLR Cholesky factorizations from Intel MKL and HiCMA Cholesky factorizations, respectively, on shared-memory systems. We compare against various Intel chip generations, i.e., Sandy Bridge, Haswell and Skylake, for both matrix kernels. Not only can HiCMA solve larger problems than Intel MKL and with a much lower slope when scaling up, the obtained performance gain is also between one and two orders of magnitude for the synthetic and the square exponential matrix kernels, respectively.

Figure 3 shows the memory footprint for various accuracy thresholds of dense (calculated) and TLR (measured) Cholesky factorizations on a million covariance matrix size from the synthetic and real world application matrix kernel, as introduced in Sect. 4. As seen in the figure, the TLR-based compression scheme exhibits more than an order of magnitude memory footprint saving with respect to naive dense Cholesky factorization from ScaLAPACK for both matrix kernels. We refer the readers to [4] for the TLR algorithmic complexity study. Furthermore, the geospatial statistics matrix kernel can not support high accuracy thresholds, since the overall matrix looses its positive definiteness. However, the fixed accuracy of 10^{-8} is used for the latter matrix kernel, as required by the original application.

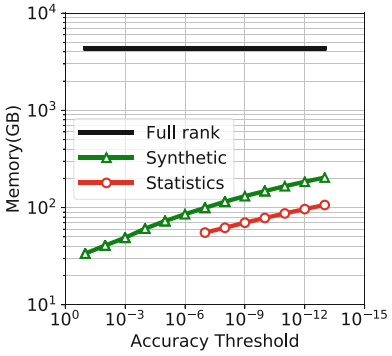


Fig. 3. Memory footprint of ScaLAPACK and hicma_dpotrf on 1M matrix size.

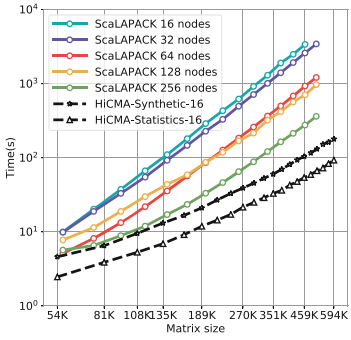
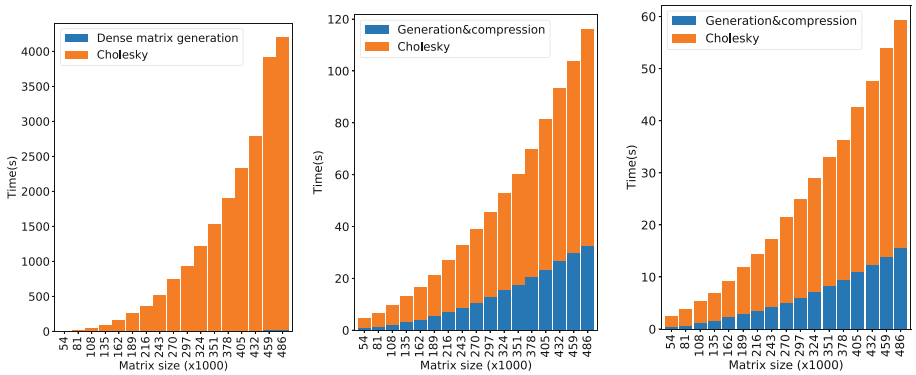


Fig. 4. Runtimes of ScaLAPACK's pdpotrf and hicma_dpotrf on *Shaheen-2*. Synthetic $\lambda = 100$.

Figure 4 shows performance comparisons of HiCMA against Intel ScaLAPACK for the TLR and dense Cholesky factorization on *Shaheen-2* distributed-memory system using both matrix kernels (including generation and compression). Since ScaLAPACK performs brute force computations, it is agnostic to the matrix kernel applications. HiCMA outperforms ScaLAPACK by using only 16 nodes as opposed to 256 nodes, up to half a million matrix size.

Figure 5 shows the time breakdown spent in generation and compression versus computation for HiCMA and ScaLAPACK Cholesky factorization on both



(a) `ScaLAPACK`. (b) `HiCMA`, synthetic. (c) `HiCMA`, statistics.

Fig. 5. Time breakdown of `ScaLAPACK`'s `pdpotrf` and `hicma_dpotrf` on *Shaheen-2* for both matrix kernels. $\lambda = 100$ for the synthetic application.

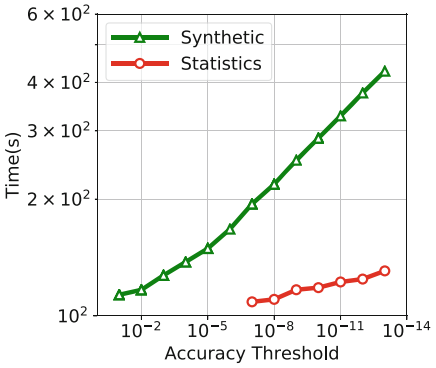


Fig. 6. Runtimes of `hicma_dpotrf` for different accuracy thresholds on 64 nodes of *Shaheen-2*. Matrix size $n = 1M$ and $nb = 2700$. Synthetic $\lambda = 200$.

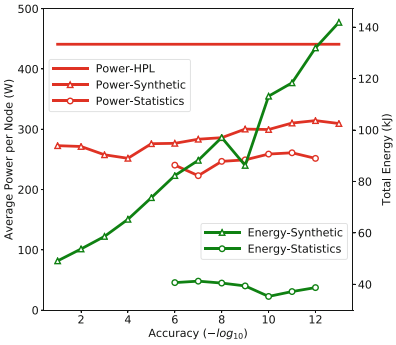
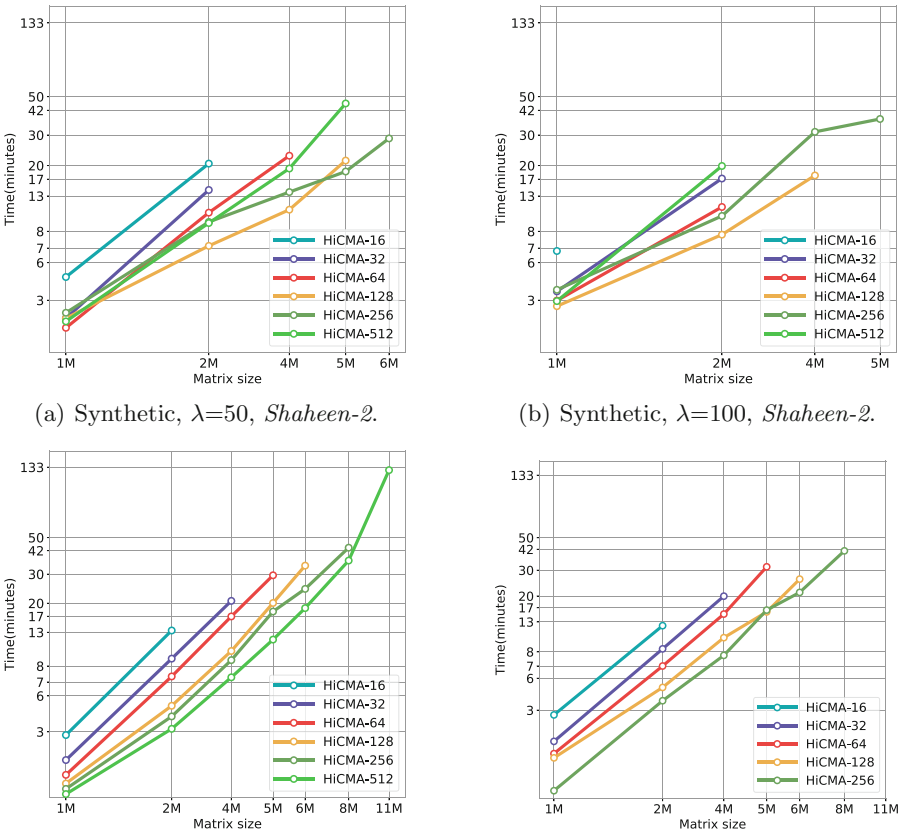


Fig. 7. Power profiling and energy consumption of `hicma_dpotrf` for different accuracy thresholds on 64 nodes of *Cray-SKL*. Matrix size $n = 1M$ and $nb = 2700$. Synthetic $\lambda = 200$.

matrix kernels. The cost of generating the dense matrix for `ScaLAPACK` is negligible compared to the computational time. However, the time to generate and compress is noticeable for `HiCMA` and counts around 25% of the elapsed time, on matrix sizes up to half a million.

Figure 6 highlights the performance impact of various accuracy thresholds for both matrix kernels. The curves have expected trends, although varying accuracy threshold does not seem to impact the performance of the square exponential matrix kernel. This is due to *maxrk*, which stays relatively the same across the accuracy thresholds.

Figure 7 show some preliminary results of power profiling and energy consumption for HiCMA TLR Cholesky factorization with various accuracy thresholds on both matrix kernels using Perftools from *Cray-SKL*. The energy consumption saving is commensurate to the performance gain in time. The recorded power offset of HiCMA compared to running HPL corresponds the under-utilized CPUs, due to low arithmetic intensity kernels.



(a) Synthetic, λ=50, *Shaheen-2*.

(b) Synthetic, λ=100, *Shaheen-2*.

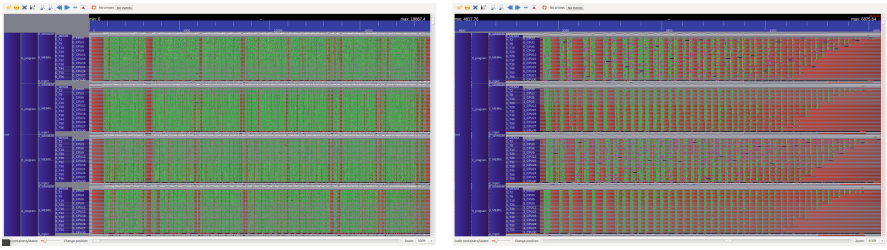
(c) Statistics, *Shaheen-2*.

(d) Statistics, *Cray-SKL*.

Fig. 8. Elapsed time of `hicma_dpotrf` for larger matrices (up to 11 million) for both matrix kernels on *Shaheen-2* and *Cray-SKL*.

Figure 8 depicts the strong scaling of HiCMA on both systems for the two matrix kernels. The synthetic matrix kernel is indeed important since it permits to show the performance bottleneck of the HiCMA's data descriptor supporting homogeneous ranks for large-scale problem sizes. Due to a large disparity of the ranks, *maxrk* has to be set to the actual maximum rank for all low-rank tiles, which engenders excessive data movement. The computation is, however,

only applied on the eligible data. For the square exponential matrix kernel, the obtained scalability is decent on both systems, considering the low arithmetic intensity of the kernels.



(a) Dense `dpotrf` time=18.120s (b) Data-sparse `dpotrf` time=1.761s

Fig. 9. Execution traces of Chameleon's `dpotrf` (a) and `hicma_dpotrf` (b) on 4 nodes of *Shaheen-2* with a matrix size of 54K. (Color figure online)

Figure 9 presents the execution traces of dense and TLR Cholesky factorizations, as implemented in task-based `Chameleon` [1] and `HiCMA`, respectively. These traces highlight the CPU idle time (red color) in `HiCMA`, since `StarPU` is not able to compensate the data movement overhead with the tasks' computations (green color). Nevertheless, there is an order of magnitude in performance between both libraries.

7 Conclusion

This paper introduces the `HiCMA` numerical library on distributed-memory systems. `HiCMA` implements a task-based tile low-rank algorithm for the Cholesky factorization. It relies on the `StarPU` dynamic runtime system to asynchronously schedule computational and communication tasks. `HiCMA` outperforms state-of-the-art dense Cholesky implementations by more than an order of magnitude in performance and saves memory footprint by the same ratio while still preserving the specific accuracy requirement of the application. The numerical robustness and high performance of `HiCMA` are demonstrated at scale using synthetic and real world matrix kernels. In particular, `HiCMA` stands as a pathfinder for approximating and effectively solving geospatial statistics applications. Future work includes using a more flexible data descriptor to better handle situations with disparate ranks, porting `HiCMA` to hardware accelerators, introducing batch processing [15] and integrating `HiCMA` into existing sparse direct solvers for the Schur complement calculations.

Data Availability Statement and Acknowledgments. The datasets and code generated during and analysed during the current study are available in the figshare repository: https://doi.org/10.6084/m9.figshare.6388202 [3] The authors would like to

thank the StarPU team at INRIA, France. This work has been partially funded by the Intel Parallel Computing Center Award. For computer time, this research used the resources from KAUST Supercomputing Laboratory for *Shaheen-2* core hours allocation.

References

1. Agullo, E., et al.: Achieving high performance on supercomputers with a sequential task-based programming model. In: IEEE TPDS (2017)
2. Agullo, E., et al.: Numerical linear algebra on emerging architectures: the PLASMA and MAGMA projects. J. Phys.: Conf. Ser. **180**, 12–37 (2009)
3. Akbudak, K., Ltaief, H., Mikhalev, A., Charara, A., Esposito, A., Keyes, D.: HiCMA (Hierarchical Computations on Manycore Architectures) library. Presented in Euro-Par 2018 paper. Figshare. Code (2018). https://doi.org/10.6084/m9.figshare.6388202
4. Akbudak, K., Ltaief, H., Mikhalev, A., Keyes, D.: Tile low rank cholesky factorization for climate/weather modeling applications on manycore architectures. In: Kunkel, J.M., Yokota, R., Balaji, P., Keyes, D. (eds.) ISC 2017. LNCS, vol. 10266, pp. 22–40. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58667-0_2
5. Ambikasaran, S., Darve, E.: An $\mathcal{O}(N \log N)$ fast direct solver for partial HSS matrices. J. Sci. Comput. **57**(3), 477–501 (2013)
6. Ambikasaran, S., Foreman-Mackey, D., Greengard, L., Hogg, D.W., O'Neil, M.: Fast direct methods for Gaussian processes. IEEE Trans. Pattern Anal. Mach. Intell. **38**(2), 252–265 (2016)
7. Amestoy, P., Ashcraft, C., Boiteau, O., Buttari, A., L'Excellent, J.Y., Weisbecker, C.: Improving multifrontal methods by means of block low-rank representations. SIAM J. Sci. Comput. **37**(3), A1451–A1474 (2015)
8. Amestoy, P.R., Duff, I.S., L'Excellent, J.Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Methods Appl. Mech. Eng. **184**(2), 501–520 (2000)
9. Aminfar, A., Ambikasaran, S., Darve, E.: A fast block low-rank dense solver with applications to finite-element matrices. J. Comput. Phys. **304**, 170–188 (2016)
10. Augonnet, C., Thibault, S., Namyst, R., Wacrenier, P.A.: StarPU: a unified platform for task scheduling on heterogeneous multicore architectures. Concurr. Comput.: Pract. Exp. **23**(2), 187–198 (2011)
11. Blackford, L.S., et al.: ScaLAPACK Users' Guide. SIAM, Philadelphia (1997)
12. Börm, S.: Efficient Numerical Methods for Non-local Operators: $\mathcal{H}^2$-Matrix Compression, Algorithms and analysis. EMS Tracts in Mathematics, vol. 14. European Mathematical Society (2010)
13. Bosilca, G., et al.: Flexible development of dense linear algebra algorithms on massively parallel architectures with DPLASMA. In: IPDPS Workshops, pp. 1432–1441. IEEE (2011)
14. Boukaram, W.H., Turkiyyah, G., Ltaief, H., Keyes, D.E.: Batched QR and SVD algorithms on GPUs with applications in hierarchical matrix compression. Parallel Comput. **74**, 19–33 (2017)
15. Charara, A., Keyes, D.E., Ltaief, H.: Tile Low-Rank GEMM Using Batched Operations on GPUs. In: Aldinucci, M., et al. (eds.) Euro-Par 2018. LNCS, vol. 11014, pp. xx–yy. Springer, Cham (2018)
16. Chiles, J.P., Delfiner, P.: Geostatistics: Modeling Spatial Uncertainty, vol. 497. Wiley, Hoboken (2009)

17. Hackbusch, W.: A sparse matrix arithmetic based on $\mathcal{H}$-matrices. part i: introduction to $\mathcal{H}$-matrices. Computing **62**(2), 89–108 (1999)
18. Hackbusch, W., Börm, S.: Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. Computing **69**(1), 1–35 (2002)
19. Hackbusch, W., Khoromskij, B., Sauter, S.: On H^2-matrices. In: Bungartz, H.J., Hoppe, R., Zenger, C. (eds.) Lectures on Applied Mathematics, pp. 9–29. Springer, Heidelberg (2000). https://doi.org/10.1007/978-3-642-59709-1_2
20. Hackbusch, W.: Hierarchical matrices: Algorithms and analysis, vol. 49. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-47324-5
21. Hackbusch, W., Börm, S., Grasedyck, L.: HLib 1.4 (1999–2012), Max-Planck-Institut, Leipzig
22. Handcock, M.S., Stein, M.L.: A Bayesian analysis of kriging. Technometrics **35**, 403–410 (1993)
23. Hénon, P., Ramet, P., Roman, J.: Pastix: a high-performance parallel direct solver for sparse symmetric positive definite systems. ParCo **28**(2), 301–321 (2002)
24. Kriemann, R.: $\mathcal{H}$-LU factorization on many-core systems. Comput. Vis. Sci. **16**(3), 105–117 (2013)
25. Kurzak, J., et al.: Designing slate: software for linear algebra targeting exascale. SLATE Working Notes 3, ICL-UT-17-06, University of Tennessee (10–2017 2017)
26. Li, X.S., Demmel, J.W.: SuperLU_DIST: a scalable distributed-memory sparse direct solver for unsymmetric linear systems. ACM TOMS **29**, 110–140 (2003)
27. Rouet, F.H., Li, X.S., Ghysels, P., Napov, A.: A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization. ACM TOMS **42**(4), 27:1–27:35 (2016)
28. SuiteSparse: A suite of sparse matrix software (2017). http://faculty.cse.tamu.edu/davis/SuiteSparse/
29. Sun, Y., Li, B., Genton, M.G.: Geostatistics for large datasets. In: Porcu, M., Montero, J.M., Schlather, M. (eds.) Space-Time Processes and Challenges Related to Environmental Problems. Lecture Notes in Statistics, vol. 207, pp. 55–77. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-17086-7_3
30. Sun, Y., Stein, M.L.: Statistically and computationally efficient estimating equations for large spatial datasets. J. Comput. Graph. Stat. **25**(1), 187–208 (2016)
31. Tyrtyshnikov, E.E.: Mosaic-skeleton approximations. Calcolo **33**(1), 47–57 (1996)