Leveraging PaRSEC Runtime Support to Tackle Challenging 3D Data-Sparse Matrix Problems

Qinglei Cao[1,6], Yu Pei[1,6], Kadir Akbudak[3,9], George Bosilca[1,7],
Hatem Ltaief[2,8], David Keyes[2,8], and Jack Dongarra[1,4,5,7]

[1]Innovative Computing Laboratory, University of Tennessee, US
[2]Extreme Computing Research Center,
Division of Computer, Electrical, and Mathematical Sciences and Engineering,
King Abdullah University of Science and Technology, KSA
[3]ASELSAN Research Center, Turkey
[4]the Oak Ridge National Laboratory, US
[5]University of Manchester, UK
[6]{*qcao3, ypei2*}*@vols.utk.edu*
[7]{*bosilca, dongarra*}*@icl.utk.edu*
[8]{*hatem.ltaief, david.keyes*}*@kaust.edu.sa*
[9]*kakbudak@aselsan.com.tr*

***Abstract*—The task-based programming model associated with dynamic runtime systems has gained popularity for challenging problems because of workload imbalance, heterogeneous resources, or extreme concurrency. During the last decade, low-rank matrix approximations—where the main idea consists of exploiting data sparsity, typically by compressing off-diagonal tiles up to an application-specific accuracy threshold—have been adopted to address the curse of dimensionality at extreme scale. In this paper, we create a bridge between the runtime and the linear algebra by communicating knowledge of the data sparsity to the runtime. We design and implement this synergistic approach with high user productivity in mind, in the context of the PaRSEC runtime system and the HiCMA numerical library. This requires extending PaRSEC with new features to integrate rank information into the dataflow so that proper decisions can be made at runtime. We focus on the tile low-rank (TLR) Cholesky factorization for solving 3D data-sparse covariance matrix problems arising in environmental applications. In particular, we employ the 3D exponential model of the Mateŕn matrix kernel, which exhibits challenging nonuniform high ranks in off-diagonal tiles. We first provide dynamic data structure management driven by a performance model to reduce extra floating-point operations. Next, we optimize the memory footprint of the application by relying on a dynamic memory allocator, and supported by a rank-aware data distribution to cope with the workload imbalance. Finally, we expose further parallelism using kernel recursive formulations to shorten the critical path. Our resulting high-performance implementation outperforms existing data-sparse TLR Cholesky factorization by up to 7-fold on a large-scale distributed-memory system, while minimizing the memory footprint up to a 44-fold factor. This multidisciplinary work highlights the need to empower runtime systems beyond their original duty of task scheduling for servicing next-generation low-rank matrix algebra libraries.**

***Index Terms*—Low-rank matrix computations, Task-based programming model, Dynamic runtime system, Asynchronous executions and load balancing, High-performance computing, User productivity, Environmental applications.**

I. Introduction

The high-performance computing (HPC) community has enjoyed an order of magnitude performance improvement every five years [1] thanks to hardware innovations and technology scaling. At the dawn of exascale, this means systems with tens of millions of concurrent threads. This rapidly evolving hardware landscape requires new software paradigms, and perhaps equally important, advanced algorithmic responsibilities [2].

The coupling of task-based programming models with dynamic runtime systems corresponds to one of the most critical paradigm shifts embraced in order to replace the traditional bulk synchronous parallel model in favor of the asynchronous execution model [3]–[5]. This versatile software solution has shown performance superiority by overlapping expensive data movement with fine-grained computations and ultimately achieving higher occupancy on the underlying hardware architecture. Moreover, runtime systems are inherently designed with the abstraction of the hardware complexity. This latter feature enhances user productivity and permits fast code deployment on massively parallel systems. At the same time, existing numerical methods have displayed limitations in addressing big data problems, due to high algorithmic complexity and large memory footprints. Low-rank matrix approximations may overcome the curse of dimensionality [6]. The main idea consists of approximating off-diagonal tiles up to an application-specific accuracy threshold and carrying out the matrix algorithm on the newly obtained data structures. This compression step may sacrifice numerical accuracy, so it results in a tunable tradeoff. By exploiting the rank structure naturally embedded in the data-sparse operator, lower complexity may be obtained in storage, data motion, and arithmetic operations, compared to traditional dense algorithms.

978-1-6654-4066-0/21/$31.00 ©2021 IEEE
DOI 10.1109/IPDPS49936.2021.00017

We propose a synergistic bridge between the runtime and linear algebra communities in the context of dealing with large-scale covariance matrices in geospatial statistics. We employ the HiCMA tile low-rank (TLR) numerical library and the PaRSEC dynamic runtime system to showcase the mutual benefits of this approach. The objective is to propagate the rank information to PaRSEC so that it can make proper runtime decisions before HiCMA operates on the computational kernels. In particular, we focus on the challenging exponential model of the Matérn matrix kernel for 3D environmental applications [7]. This model results in heterogeneous rank distribution with high-rank tiles located outside of the diagonal tiles. We extend PaRSEC with new functionality that takes into account the rank information in the task dataflow: (1) a dynamic data structure management driven by a performance model to reduce extra floating-point operations; (2) a dynamic memory allocator to further optimize memory footprint; (3) a rank-aware data distribution to cope with the workload imbalance; and (4) a recursive formulation of computational kernels to expose concurrency during the critical path. We use these features to leverage the performance of TLR Cholesky factorization at the heart of the maximum likelihood estimation (MLE) [8]. MLE is employed for estimating parameters and reaches very high dimensions in 3D environmental applications.

The resulting TLR Cholesky from HiCMA powered by PaRSEC (called PaRSEC-HiCMA-New) outperforms a previous implementation (called PaRSEC-HiCMA-Prev) by up to a 7-fold speedup on a large-scale distributed-memory system, while minimizing the memory footprint up to a 44-fold factor. We believe this multidisciplinary symbiosis is fundamental to porting the next-generation of low-rank matrix algebra libraries to exascale. This demands empowering runtime systems beyond their original duty of task scheduling.

The remainder of this paper is as follows. Section II presents related work and lists our contributions. Section III provides background information about covariance matrix problems for 3D environmental applications and describes how HiCMA and PaRSEC synergistically solve them. In particular, Section IV details the challenges carried by the 3D exponential kernels. Section V introduces the PaRSEC dynamic data structure management, assisted by a performance model, that requires new numerical kernel developments in HiCMA, as explained in Section VI. Section VII highlights the novel rank-aware runtime optimizations integrated in PaRSEC. We report performance results in Section VIII and conclude.

II. Related Work

The richness of the recent literature on low-rank matrix approximations is evidence of a compelling new approach to big data scientific problems [9]. In particular, hierarchical matrices ($\mathcal{H}$-matrices) [10]–[12] constitute a family of blockwise low-rank matrix approximations used to reduce the arithmetic complexity and the memory footprint. Depending on the data sparsity pattern of the operator, there exist many $\mathcal{H}$-matrix data compression formats for weak admissibility (e.g., Hierarchically Semi-Separable (HSS) [13], [14], Hierarchically Off-Diagonal Low-Rank (HODLR) [15]) and strong/standard admissibility (e.g., $\mathcal{H}^2$-matrix [16], Block/Tile Low-Rank (BLR / TLR) [2], [17]). Thanks to their inherent recursive formulations, both compression formats may attain linear arithmetic complexity and memory storage for some matrix problems and operations [18]. Weak admissibility is well suited for off-diagonal blocks exhibiting low ranks (e.g., typically 2D problems), while strong admissibility can still maintain the lower complexity in the presence of off-diagonal blocks with high ranks (e.g., typically exacerbated in 3D).

While the theoretical lower bounds of these low-rank approximation schemes are attractive—replacing factors of problem dimension with factors of maximum block rank away from the dense diagonal blocks—their deployment on massively parallel systems has exposed their limitations, especially in problems where maximum block rank is high. The recursive formulations required to exploit low rank hinder the overall performance due to a low hardware occupancy exacerbated by the excessive synchronization. Flattening the recursion tree and avoiding synchronizations in-between hierarchical steps can mitigate this inefficiency. They highlight the impact of batch executions on graphics processing units (GPUs) to increase the hardware occupancy [2], [19], [20] for iterative solvers. They employ the HiCMA task-based numerical library with the StarPU dynamic runtime system [8], [21] for attenuating load imbalance effects on distributed-memory systems in the context of TLR Cholesky factorizations. Further performance improvement of HiCMA has been obtained using the hybrid data distribution implemented in the PaRSEC dynamic runtime system [22], [23]. TLR provides a particularly nice trade-off between optimality, performance, and user productivity [24] since traditional dense tile algorithms can be used.

Our contributions are as follows. We leverage the PaRSEC support for data-sparse matrix computations by embedding in the dataflow at runtime a new dynamic data structure management driven by a performance model to reduce extra floating-point operations. This requires implementing new numerical kernels in HiCMA to enable the resulting TLR Cholesky algorithm at scale. We develop a dynamic memory allocator to further optimize memory footprint, breaking with the traditional, rigid data descriptor from ScaLAPACK. We then provide a rank-aware data distribution to better balance the workload, and further expose concurrency to shorten the critical path by integrating a recursive formulation of all dense computational kernels. We demonstrate on a 3D exponential matrix kernel that engenders high-rank heterogeneity, illustrating performance improvement of the TLR Cholesky on up to 12 million spatial locations for the MLE when simulating large-scale environmental applications.

III. Bridging the Gap Between Linear Algebra and Runtime Communities

This section provides background information on a statistical model used for climate and weather prediction applications,

recalls the TLR Cholesky factorization in HiCMA, describes the PaRSEC dynamic runtime system, and lays out the algorithmic and software foundation roadmap for next-generation computational linear algebra libraries.

A. Geospatial Statistics Application

Geospatial statistics applications are typically data-sparse problems that can be modeled with the MLE-based iterative optimization procedure as follows:

$$\ell(\boldsymbol{\theta}) = -\frac{n}{2}\log(2\pi) - \frac{1}{2}\log|\boldsymbol{\Sigma}(\boldsymbol{\theta})| - \frac{1}{2}\mathbf{Z}^\top\boldsymbol{\Sigma}(\boldsymbol{\theta})^{-1}\mathbf{Z}, \quad (1)$$

where the covariance matrix $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ is symmetric and positive-definite, containing the correlations between n geospatial locations; $\mathbf{Z}$ represents the vector of measurements; and $\boldsymbol{\theta} = (\theta_1, \theta_2, \theta_3)^\top$ is the model parameter vector to optimize. The objective is to calculate $\widehat{\boldsymbol{\theta}}$, which represents the MLE of $\boldsymbol{\theta}$ in Equation (1). The Matérn function defines each entry of $\boldsymbol{\Sigma}(\boldsymbol{\theta})$:

$$C(r;\boldsymbol{\theta}) = \frac{\theta_1}{2^{\theta_3-1}\Gamma(\theta_3)}\left(\frac{r}{\theta_2}\right)^{\theta_3}\mathcal{K}_{\theta_3}\left(\frac{r}{\theta_2}\right), \quad (2)$$

where $r = \|\mathbf{s} - \mathbf{s}'\|$ is the distance between any two spatial locations and $\mathcal{K}_{\theta_3}$ denotes the modified Bessel function of the second kind of order θ_3. The Matérn kernel is also used in machine learning [25] and image processing [26].

As n increases, the cubic algorithmic complexity renders solving the evaluation of MLE burdensome. Fortunately, covariance matrices are usually hierarchically low-rank and may be sped up with low-rank matrix approximations.

B. The HiCMA Library

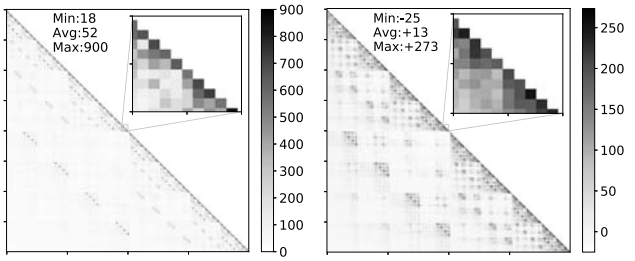
The HiCMA numerical library includes TLR matrix computations that exploit the data sparsity of the covariance matrix. HiCMA [2], [21] relies on STARS-H (https://github.com/ecrc/stars-h) to generate the covariance matrix problem and compress each off-diagonal tile up to an application-dependent accuracy threshold. The dense representation translates into a tile-centric compressed representation that captures the most significant singular values (i.e., the rank of the tile). This compressed data structure is composed of two tall-and-skinny matrices per tile—i.e., U and V of size b $\times$ k—with b the tile size and k the rank. Since tiles may have different ranks, HiCMA consolidates the rank heterogeneity by using a unique maxrank to define a homogeneous data descriptor at the cost of a higher memory footprint. HiCMA follows the traditional two-dimensional block cyclic data distribution (2DBCDD) from ScaLAPACK that requires uniform block sizes. We set the upper limit for maxrank to b/2 to maintain the competitiveness of low-rank matrix approximations over dense matrix computations, as far as the memory footprint is concerned. This situation may be suboptimal for two reasons. The presence of a single high rank (e.g., $k \sim$ b/2) will define the actual maxrank for all off-diagonal tiles, which may jeopardize the benefits of TLR compression ratio. Further, the presence of several high ranks may also increase the overall arithmetic complexity. Therefore, the sensitivity to rank distribution of the rigid data descriptor may hinder the overall performance. Once the matrix is compressed, HiCMA can then operate on the low-rank representation of the matrix. It performs the standard matrix operations based on high-performance kernel implementations specifically designed for manipulating the underlying TLR data compression format. HiCMA currently supports StarPU, OpenMP, and PaRSEC runtime systems to orchestrate the task scheduling of matrix computations.

C. The PaRSEC Runtime System

PaRSEC [27] is a task-based runtime for distributed heterogeneous architectures capable of dynamically unfolding description of a directed acyclic graph (DAG) of tasks on a set of resources. PaRSEC tracks all data dependencies by efficiently shepherding data between memory spaces (between nodes but also between different memories on different devices) and schedules tasks across heterogeneous resources. *Starvation*, *latency*, *overhead*, and *heterogeneity* are the four main barriers on which PaRSEC focuses to overcome algorithm scalability and efficiency. Domain-specific language (DSLs) are utilized to allow domain experts to ignore the underlying complexity of implementation, which relies on a dataflow model to create dependencies between tasks and targets the expression of maximal parallelism. Parameterized task graph (PTG) [28] and dynamic task discovery (DTD) [29] are two representative DSLs in PaRSEC. PTG, used in this paper, uses a concise, parameterized task-graph description known as job data flow (JDF) to represent dependencies between tasks. Collective communications are fully supported in PTG to enhance application developer productivity, which distinguishes PaRSEC in task-based runtime systems. For instance, the collective communication in StarPU is limited, assuming all dependencies related to a collective communication need to be discovered when collective communication is performed [30].

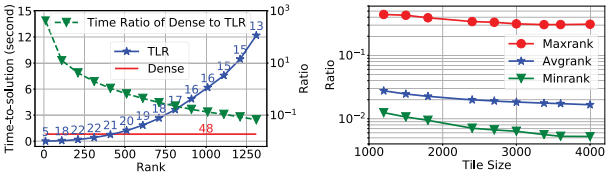
D. A Renaissance *in Computational Linear Algebra*

We bridge the linear algebra and runtime communities using a synergistic approach that combines knowledge expertise of HiCMA and PaRSEC. This software solution permits not only alleviating performance bottlenecks but also increasing productivity when dealing with large-scale applications on distributed-memory systems. This is possible by bringing awareness of the rank information right after the compression to PaRSEC before HiCMA takes over. This critical insight may not be useful when block sizes are identical, as in traditional dense linear algebra (e.g., ScaLAPACK). However, with low-rank matrix approximations, HiCMA can leverage the PaRSEC support beyond the usual task scheduling to additionally handle the impact of rank heterogeneity on the process grid, the data distribution, the data movement, and the load balancing. PaRSEC is able to abstract the hardware complexity as well as the challenges in dealing with the complex low-rank matrix algorithms. Numerical developers of low-rank matrix algorithms can focus more on the optimality of their sequential code before empowering runtime systems on massively parallel systems. This separation of concerns is enabling a *Renaissance* in computational linear algebra.



(a) Initial ranks. (b) Rank variations.

Fig. 1: Rank distributions for off-diagonal tiles with a matrix size $\mathtt{N} = 1.08$M, tile size $\mathtt{b} = 2700$ and the number of tiles $\mathtt{NT} = 400$. The heat map in (a) shows the initial ranks after compression and the heat map in (b) shows the difference between the ranks before and after the TLR Cholesky factorization. Each point in a 400-by-400 heat map corresponds to a tile in the matrix and the color scale of the point represents the rank of the tile.



(a) TLR GEMM vs dense GEMM. (b) Ratio: rank to tile size.

Fig. 2: (a) Single-core dense and TLR GEMM performance with $\mathtt{b} = 2700$; the sustained performance (Gflop/s) is labeled with blue for TLR and red for dense. (b) Ratio of maxrank, avgrank and minrank to tile size with $\mathtt{N} = 1.08$M.

IV. New Challenges with 3D Exponential Kernels

In our examples, we set the variance parameter $\theta_1 = 1.0$, the correlation parameter $\theta_2 = 0.1$, and the smoothness parameter $\theta_3 = 0.5$. This setting reduces the Matérn kernel from Equation 2 to the decaying 3D exponential kernel $C(r;\boldsymbol{\theta}) = \exp(-r/0.1)$ (**st-3D-exp**). This matrix kernel variant is suited to model medium correlations and rough fields (e.g., estimating wind speed or temperature changes with altitude). We use Morton ordering [31] for a good compression ratio. Compared to previous works [22], [23], **st-3D-exp** presents new challenges with higher ranks observed after compression due to the medium correlation and the rough field. The heat maps in Fig. 1 display the rank distribution for off-diagonal tiles. In particular, the first figure shows the initial rank distribution (i.e., after compression) along with the minimum, average, and maximum rank (minrank, avgrank and maxrank respectively). The second figure shows the rank variations before and after the factorization. With the zoom-in on tiles close to the diagonal, it is clear that the rank heterogeneity is more pronounced with **st-3D-exp** than previously studied matrix kernels (see Fig. 2 of [23]) and becomes even higher after the factorization. From these figures, we track the following ratios to control the overall arithmetic complexity and memory footprint:

- $\mathtt{ratio\_maxrank} = \frac{\mathtt{maxrank}}{\mathtt{b}}$
- $\mathtt{ratio\_discrepancy} = \frac{\mathtt{maxrank-minrank}}{\mathtt{b}}$

where b is the tile size, and they are only known at runtime after the compression step and need to be escalated to the runtime for proper internal usage.

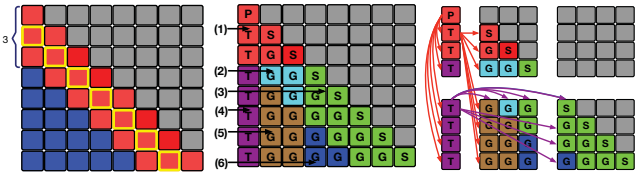
Fig. 2(a) reports the time-to-solution (left y-axis) as well as its ratio (right y-axis) of dense GEMM to TLR GEMM on a single core. These kernels are the most time-consuming operations in TLR and dense Cholesky. TLR GEMM can be more expensive than dense GEMM when the rank exceeds a threshold, determined by the arithmetic complexities of TLR and dense GEMMs. Also, the gap between TLR and dense GEMM widens as rank continues to rise. The figure also annotates the kernel performances in Gflops/s of TLR and dense GEMM. The TLR GEMM performance falls in between the regime of memory-bound and compute-bound, achieving roughly $1/3$ of dense GEMM. TLR GEMM performance tapers off at both ends of rank. When the rank is small, TLR GEMM is mostly memory-bound with lower performance. As rank increases, it becomes more compute-bound and achieves higher performance. However, when the rank continues to grow, the expensive recompression step in the TLR GEMM kernel dominates, and the performance starts decreasing. The detailed explanation about internal steps of the TLR GEMM operation can be found in Section 8.1 of [32]. Fig. 2(b) depicts the impact of tile size on the rank information (i.e., maxrank, avgrank and minrank) after compressing a matrix of size $\mathtt{N} = 1.08$M. As tile size increases, the overall trend for rank goes down, which indicates a higher data sparsity attained due to the medium correlation. At the same time, a large tile size reduces the degree of parallelism, while a small tile size leads to high ratio_maxrank and ratio_discrepancy.

All in all, **st-3D-exp** creates a new level of productivity, performance, and scalability challenges for HiCMA that may only be addressed by a versatile runtime support from PaRSEC, as explained in subsequent sections.

V. Dynamic Data Structure Management

A. The Necessity to Densify the Matrix Operator

As mentioned heretofore, a tile's rank increases when approaching the diagonal, leading to a TLR GEMM operation more expensive than dense GEMM for **st-3D-exp**. This may delay the critical path (POTRF as well as the first TRSM and SYRK for each panel factorization) and ultimately impact the overall time to solution. The idea is to have PaRSEC dynamically manage the flavor of the data structure at runtime. PaRSEC detects these specific tiles with high ranks at runtime and triggers, only for those, a rollback to the original dense format. Fig. 3(a) shows the symmetric tile matrix (only the lower triangular part is referenced) with a mixture of TLR (blue) and dense (red) data layouts. This BAND-DENSE-TLR Cholesky factorization engenders a different work-flow compared to the regular factorization, since it needs to take into account



(a) Data layout view. (b) Compute kernels. (c) Data flow.

Fig. 3: Data layout view, numerical kernels and data-flow of BAND-DENSE-TLR Cholesky with BAND_SIZE = 3. In figures, P means POTRF; T, TRSM; S, SYRK; G, GEMM. (a) Red means dense tiles and blue means low-rank tiles. (b) Unrolling the first panel factorization with colors representing different kernels. (c) The red arrows show data-flow with dense tiles while the purple arrows data-flow with low-rank tiles.

the coexistence of both data formats during the data-flow with new supportive computational kernels from HiCMA. This densification process of the matrix operator eventually proves superior in performance and is driven by a performance model assisted by an auto-tuner.

B. Performance Model Based on BAND_SIZE Auto-Tuning

We introduce a performance model that acts as an auto-tuner for identifying the BAND_SIZE parameter that controls the number of sub-diagonals that are required to roll back to dense formats. For instance, Fig. 3(a) shows the main tile diagonal with BAND_ID = 1 and highlights in yellow the tile sub-diagonal with BAND_ID = 2. BAND_SIZE is an inherent tunable parameter of the BAND-DENSE-TLR Cholesky algorithm and may vary depending on the studied covariance matrix problems. Algorithm 1 presents the performance model to minimize the total number of floating-point operations (FLOPs) by auto-tuning the BAND_SIZE parameter. Implemented in PaRSEC, the tuning procedure selects a suitable BAND_SIZE automatically, based on the initial rank distribution revealed right after the matrix compression. An artificial distribution of one-dimensional block-cyclic data distribution (1DBCDD) is provided to evenly distribute all tiles in a sub-diagonal to all processes, so that all resources are utilized to speed up the progress. Once BAND_SIZE is calculated, the tiles in sub-diagonals with BAND_ID < BAND_SIZE are translated back to dense format in a transparent manner to users. Currently, we do not support the case in which the ranks may grow or shrink during the factorization, since it is hard to predict in advance. However, one can imagine an adaptive online auto-tuning that densifies or sparsifies the tiles on-demand, but this is beyond the scope of this paper. This runtime auto-tuning procedure enables PaRSEC to leverage HiCMA toward a wider coverage of 3D data-sparse covariance matrix problems beyond **st-3D-exp** studied herein.

Algorithm 1: Algorithm for BAND_SIZE auto-tuning.

Input : Matrix data descriptor
Generate the matrix with BAND_SIZE = 1
Globalize the rank distribution to all the processes
Set $ID = 1$ and initialize $fluctuation$
do
  $ID := ID + 1$
  ops_dense = total TRSM and GEMM FLOPs of all tiles in sub-diagonal with BAND_ID = ID if executing in dense format
  ops_tlr = total TRSM and GEMM FLOPs of all tiles in sub-diagonal with BAND_ID = ID if executing in low-rank format
while $ops\_dense < fluctuation \times ops\_tlr$;
Output: BAND_SIZE = $ID - 1$

VI. New Kernel Implementations in HiCMA

Densifying the matrix operator requires the implementation of new computational kernels in HiCMA. Fig. 3(b) distinguishes six tile regions that group kernels with the same data layout property. With the combination of regions ((1)–(6)) and kernels (POTRF, TRSM, SYRK and GEMM), there are ten types of different kernels involved, codenamed as "(region)-kernel." Each kernel is briefly described below.

- (1)-POTRF: Cholesky factorization of a diagonal (lower triangular) tile as in LAPACKE.
- (1)-TRSM: dense triangular solve as in CBLAS.
- (4)-TRSM: low-rank triangular solve as in HCORE_DTRSM [32].
- (1)-SYRK: dense symmetric rank-k update as in CBLAS.
- (3)-SYRK: low-rank symmetric rank-k update as in HCORE_DSYRK [32].
- (1)-GEMM: dense matrix-matrix multiplication as in CBLAS, $C = C - A \times B^T$.
- **(2)-GEMM: modified dense matrix-matrix multiplication,** $C = C - U_A \times V_A^T \times B^T$**.**
- **(3)-GEMM: modified dense matrix-matrix multiplication,** $C = C - U_A \times V_A^T \times V_B \times U_B^T$**.**
- **(5)-GEMM: modified low-rank matrix-matrix multiplication,** $U_C \times V_C^T = U_C \times V_C^T - U_A \times V_A^T \times B^T$**.**
- (6)-GEMM: low-rank matrix-matrix multiplication as in HCORE_DGEMM [32], $U_C \times V_C^T = U_C \times V_C^T - U_A \times V_A^T \times V_B \times U_B^T$.

TABLE I: Arithmetic complexity of all kernels.

ID	(Group)-Name	Complexity
0	(1)-POTRF	$\frac{1}{3} \times b^3$
1	(1)-TRSM	b^3
2	(4)-TRSM	$b^2 \times b$
3	(1)-SYRK	b^3
4	(3)-SYRK	$2 \times b^2 \times k + 4 \times b \times k^2$
5	(1)-GEMM	$2 \times b^3$
6	(2)-GEMM	$4 \times b^2 \times k$
7	(3)-GEMM	$2 \times b^2 \times k + 4 \times b \times k^2$
8	(5)-GEMM	$34 \times b \times k^2 + 157 \times k^3$
9	(6)-GEMM	$36 \times b \times k^2 + 157 \times k^3$

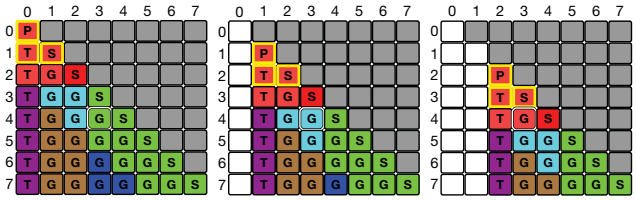
Compared to the previous HiCMA kernels [32], three new kernels (**in bold**) have been implemented to handle the workflow of BAND-DENSE-TLR. The arithmetic complexities of all kernels are reported in Table I (b represents the tile size, and k is the rank of that tile). The kernel group (1) with $O(b^3)$ complexity is the most expensive and usually operate on dense tiles close to the diagonal. The other remaining kernel groups have lower complexity and run on compressed tiles.

VII. Novel Rank-Aware Optimizations in PaRSEC

A. Dataflow Runtime Adaptation

We classify the resulting dataflow into two categories: LOCAL (connecting tasks on the same process, including SYRK → SYRK, SYRK → POTRF, GEMM → GEMM and GEMM → TRSM) and REMOTE (connecting tasks on different processes, including POTRF → TRSM, TRSM → SYRK and TRSM → GEMM). Only the REMOTE dataflow can post communications. Fig. 3(c) depicts REMOTE dataflow within a panel factorization, including three broadcast communications—POTRF to TRSM, TRSM to GEMM in a row, and TRSM to GEMM in a column—and one peer-to-peer communication, TRSM to SYRK. This figure highlights the broadcast from POTRF to TRSM and two kinds of broadcasts from TRSM to GEMM with arrows of different colors representing different types of data encapsulated in dataflow—red meaning dense data while purple low-rank data. The dynamic data structure that supports BAND-DENSE-TLR pressures the runtime to accommodate data motion with heterogeneous data layout.

Fig. 4 shows the BAND-DENSE-TLR algorithm by highlighting the first three panel factorization steps with NT $= 8$ (the number of tiles in a dimension) and BAND_SIZE $= 3$. Different colors represent kernel regions or completion, similar to Fig. 3. Tiles with bold yellow boundaries are included in the critical path for that panel factorization, assuming the critical path spans distance 1 in the dataflow dependencies. It is worth noting that the type of kernels using a specific tile will change across successive iterations; for instance, (3)-GEMM, (2)-GEMM, and (1)-GEMM are successively called on the tile with index $(m, n) = (4, 3)$ for the first three iterations. This represents one example of the levels of complexity that PaRSEC abstracts from HiCMA's TLR algorithms.



(a) Panel factorization of the 1st iteration. (b) Panel factorization of the 2nd iteration. (c) Panel factorization of the 3rd iteration.

Fig. 4: BAND-DENSE-TLR Cholesky algorithm. Colors represent different tile regions, with white labeling the completed task. The numbers are tiles' row and column index. P means POTRF; T, TRSM; S, SYRK; G, GEMM.

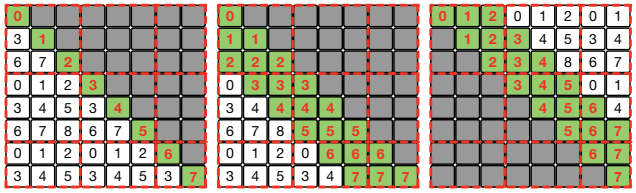
B. Dynamic Memory Designation

As mentioned above, **st-3D-exp** shows properties of high ratio_maxrank, high ratio_discrepancy, and rank variations during the Cholesky factorization, even when the BAND-DENSE-TLR feature is in use. If the memory is statically allocated based on the pre-defined maxrank parameter (see Section III-B), it will (1) limit the problem size that can be solved on a specific set of computational resources, and (2) restrict the tile size used to expose parallelism. Indeed, as indicated in Sections III-B and IV, there is an inverse relationship between the accuracy of the low rank representation and the tile size used, in the sense that the maxrank of a tile increases as the tile size decreases—a problem that can further be exacerbated by the possible rank growth during the factorization. The PaRSEC runtime system provides users a flexible way to designate the input data for a task, but also the data the task will propagate to its successors. This capability allows the user code to precisely allocate the needed memory and is then injected back into the runtime to serve its purpose.

PaRSEC-HiCMA-New takes advantage of this capability in two distinct ways. First, during the initialization, the matrix is allocated based on the initial rank according to the required accuracy threshold using temporary memory from a reusable memory pool provided by the PaRSEC runtime. Once the matrix is generated, the actual rank of each tile becomes known, and the exact amount of memory necessary for each tile can be allocated and associated with the corresponding constructs in the runtime system. Second, during the factorization itself, the rank of the tiles might change. To adapt to this change, the low-rank GEMM kernels, (5)-GEMM and (6)-GEMM, are split into two stages clearly delimited by the recompression operation. The first stage consists of operations until recompression and the second stage consists of the remaining operations after recompression. As a result, the memory for each tile can not only be reallocated but also reassociated with the runtime system between these two stages if rank growth occurs as a result of re-compression. This reassociation takes advantage of the distinction between logical and physical data in the PaRSEC runtime. The algorithm is free to reassess at any moment the association between logical and physical, and the runtime will automatically adapt (both in terms of dependencies between tasks and data movements). This simple but extremely useful feature is one of the most critical differences between the PaRSEC runtime and other task-based runtime, and is one of the key components that allowed our approach to scale to unprecedented problem sizes.

C. Hybrid Data Distribution

The discrepancy between dense on-band tiles and compressed off-band tiles can be expressed using 3 types of metrics: memory, computation, and communication. (1) Memory: memory needed for off-band tiles is proportional to their rank, and they require $2bk$ elements, while on-band tiles require b^2 elements. (2) Computation: after rolling tiles with high rank back into the dense format according to the arithmetic complexity from Table I, ranks of the remaining compressed



(a) (b) (c)

Fig. 5: Hybrid data distribution suitable for BAND-DENSE-TLR Cholesky and the corresponding process ID. (a) as used in PaRSEC-HiCMA-Prev with BAND_SIZE = 1; (b) for lower triangular matrix and (c) for upper triangular matrix, both with BAND_SIZE = 3.

tiles are more than an order of magnitude smaller than the tile size. Thus, according to Table I, kernels operating on dense tiles have a higher computational cost than the corresponding kernels on compressed tiles. (3) Communication: tasks operating on on-band tiles send dense data (b^2 elements) while those operating on off-band tiles send compressed data ($2bk$ elements) (see Fig. 3(c)).

PaRSEC-HiCMA-Prev [22] introduces the concept of "band distribution," which superposes two intertwined 2DBCDDs using different process grids, but restricts its usage to a band of size 1, as shown in Fig. 5(a), where only the diagonal tiles are dense and evenly distributed across all processes in a 1DBCDD. In BAND-DENSE-TLR algorithm, BAND_SIZE could be bigger than one but remains relatively small compared to NT ($1 <$ BAND_SIZE $\ll$ NT, demonstrated in Section VIII), thus tiles on-band (not only diagonal) need to be evenly distributed across all processes to address any imbalance issues described thus far. We propose to adapt this "band distribution" to the problem type as shown in Fig. 5(b) for the lower triangular matrix and Fig. 5(c) for the upper triangular matrix. Distribution on the band could be seen as a modified 1DBCDD: row-based (tiles on-band in a row mapped to the same process) for the lower triangular matrix, and column-based (tiles on-band in a column mapped to the same process) for the upper triangular matrix. The main reasons behind such a choice are twofold: a well balanced panel factorization because dense TRSMs in the panel factorization are distributed to different processes and can therefore be executed in parallel; and the reduction of communications on the critical path because kernels on the tiles on the same row are mostly sequential, and the distribution chosen here removes the need for communications between these kernels.

D. Recursive Numerical Kernels

Tasks on and near the critical path are important because they affect the time to solution by impacting the discovery of the next panel factorization (i.e., the lookahead). In BAND-DENSE-TLR Cholesky, we can consider that the entire band, composed only by dense tiles, (red region in Fig. 4), and therefore performing only traditional dense Level-3 BLAS kernels is our critical path at distance BAND_SIZE. Therefore, these tasks need to be promoted and executed as quickly as possible, to enable all available parallelism in the off-band part. Moreover, speeding them up will reduce the waiting time by minimizing the potential starvation—particularly at the end of the execution where the opportunities for parallelism are less. PaRSEC-HiCMA-Prev [22] utilizes the concept of "nested computing" to expedite POTRF execution by recursively dividing the local computations on large dense tiles into smaller kernels. The direct outcome is more parallelism, which can then be exploited if computational resources are available. In PaRSEC-HiCMA-New, we extend this idea, not only targeting POTRF but instead applying to all kernels in region (1) of Fig. 3(b) including (1)-POTRF, (1)-TRSM, (1)-SYRK and (1)-GEMM. As a result, all dense kernels close to the critical path can potentially be sped up, such that the discovery of the next panel factorization is expedited with the possibility of increasing utilization of hardware resources.

VIII. Performance Results and Analysis

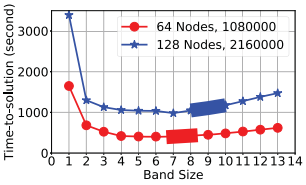
A. Environment Settings

The experiments were conducted on Shaheen II at KAUST, a Cray XC40 system with 6,174 compute nodes, each of which has two 16-core Intel Haswell CPUs running at 2.30 GHz and 128 GB of DDR4 main memory. Intel compiler suite 19.0.5.281 with sequential Math Kernel Library (MKL) version 2019.5 for optimized BLAS and LAPACK kernels is deployed. Numerical backward errors have been consistently validated against the application accuracy threshold to ensure correctness. We compress off-band tiles and retain their most significant singular values (and associated vectors) above the accuracy threshold of 10^{-8} (except in Section VIII-G), which ultimately yields an absolute numerical error of order 10^{-9} in the solution of the linear system in Equation 1 to make it consistent as in [22]. This 10^{-9} tolerance is sufficient to satisfy the prediction accuracy requirements of the 3D climate and weather prediction applications, as described in [8]. We employ the "band distribution" and a 2DBCDD for tiles off-band with a process grid $P \times Q$ (as square as possible) where $P \leq Q$. We use the same BAND_SIZE for BAND-DENSE-TLR algorithm and "band distribution". Calculations and communications are performed in double-precision floating-point arithmetic. We run our experiments at least three times; and since no major performance variability has been noticed, the minimum time to solution is reported.

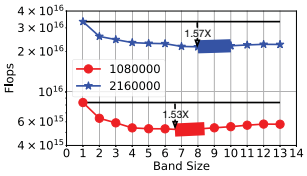
B. Impact of BAND_SIZE Auto-Tuning

The BAND_SIZE parameter for BAND-DENSE-TLR algorithm is automatically tuned, in a process totally transparent to the user. The autotuning process includes (1) generating matrix with BAND_SIZE = 1, (2) BAND_SIZE auto-tuning, and (3) matrix regeneration for tiles within a band with the tuned BAND_SIZE. Fig. 6 evaluates the entire process of BAND_SIZE auto-tuning on two settings N = 1.08M and N = 2.16M.

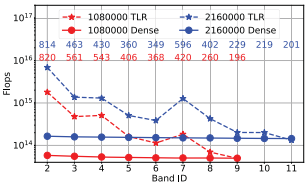
- Fig. 6(a) shows changes in time-to-solution and Fig. 6(b) the corresponding total flops while varying the



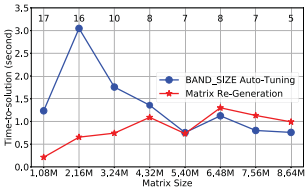
(a) Time to Solution.



(b) Total flops.



(c) Flops per sub-diagonal.

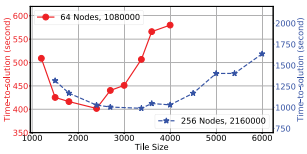


(d) Evaluation on 512 nodes.

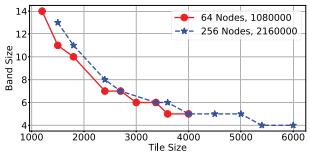
Fig. 6: Evaluation of BAND_SIZE auto-tuning.

BAND_SIZE, and both with b = 2400. The rectangle boxes are range with $fluctuation \in [0.67, 1]$ (see Algorithm 1). We choose to use the minimum value of this range in the remaining experiments because of (1) FLOPs increase in TRSM and SYRK near the critical path if rolling back the tiles to dense format, (2) rank variations during factorization, and (3) significance of dense-band dominating time-to-solution (demonstrated in Section VIII-F). It is clearly visible in these figures that each case has a sweet spot in terms of time-to-solution and the corresponding FLOPs, and that the predicted BAND_SIZE is close to the optimal.

- Fig. 6(c) demonstrates the process of Algorithm 1 with $fluctuation = 1$ by comparing the FLOPs of each sub-diagonal in dense and TLR format. Annotations in this figure are the maxrank for the corresponding sub-diagonal, for the two matrix sizes we investigated (red for N = 1.08M and blue for N = 2.16M). For the TLR format, the FLOPs of a sub-diagonal decrease as BAND_ID increases mainly because of the reduction in maxrank, but also due to the reduction of the number of tiles in the sub-diagonal, and therefore the number of operations on these tiles. This second reduction is also true for the dense format; successive sub-diagonals have a monotonically decreasing number of tiles and thus marginally fewer FLOPs.
- Fig. 6(d) shows the time-to-solution of the BAND_SIZE auto-tuning process and the cost of the matrix regeneration after BAND_SIZE tuning for the experiments on 512 nodes. The corresponding tuned BAND_SIZE is marked at the top of the figure. Based on these results it is clear that the time of BAND_SIZE auto-tuning process, as well as the necessary time for the matrix regeneration, is negligible when compared to the cost of the entire Cholesky factorization.



(a) Time to solution.



(b) Corresponding band size.

Fig. 7: Effect of tile size.

C. Suitable Tile Size Selection

Tile size is also a critical parameter for the tile-based algorithm, trading off between performance per task and concurrency between tasks. A more in-depth analysis of the performance/concurrency tradeoff for regular problems (where the cost of all tasks is similar) can be found in [33] for multiple runtime systems. For TLR Cholesky, [23] proposed a model to calculate the approximate optimal tile size by assuming a first-order approximation—the serial part (the critical path in the algorithm) at distance one overlapping with the parallel part (everything outside the critical path). This assumption does not hold for the **st-3D-exp** application because of the higher ranks in the matrix, but also due to the dense operations in the band. Putting aside the dense band part, [17] proposed that the minimal operations count could be attained by a TLR matrix computation when $b = O(\sqrt{N})$, which we can use as a rough starting point. Fig. 7(a) present two experiments, the time-to-solution for a TLR factorizations using different tile sizes on 64 nodes with a N = 1.08M (left y-axis) and 256 nodes with a N = 2.16M (right y-axis). The estimated value computed using [17], around 1039 for the red line and 1469 for the blue, are reasonably good estimates as a starting point. Fig. 7(b) displays the corresponding auto-tuned BAND_SIZE which decreases as tile size increases. This can be explained from the observation in Fig. 2(b) that as tile size increases, the ratio_maxrank decreases, thus reducing the need to convert compressed tiles into dense tiles. In fact, a suitable tile size depends on many factors: the data and the algorithm, the compute capabilities of the computing resources, the network performance and capabilities, matrix size, etc. Proposing a model to predict the optimal tile size for a complicated hybrid BAND-DENSE-TLR algorithm is outside the scope of this paper. However, for the scope of our study it is enough to conduct the experiments starting from a tile size (such as the one proposed in [17]), and stopping when the time-to-solution trend changes, basically finding a local minima.

D. Impact of Dynamic Memory Designation

PaRSEC-HiCMA-New can allocate the exact memory amount for each tile based on the actual rank during factorization to remove the restriction imposed by the statically predefined maxrank in PaRSEC-HiCMA-Prev. Fig. 8 evaluates this feature for a case running on 512 nodes.

The left figure displays the memory reduction between allocating each compressed tile as $2 \times \text{maxrank} \times b$ in PaRSEC-HiCMA-Prev and as $2 \times k \times b + r$ in

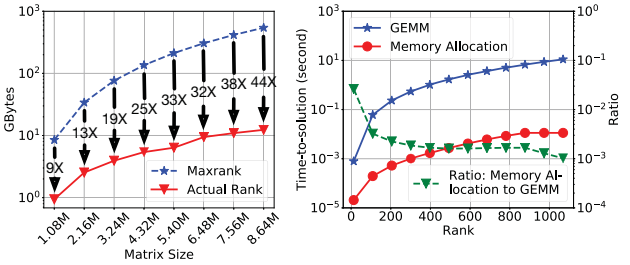


Fig. 8: Memory usage evaluation. Left, memory reduction on 512 nodes; right, analysis of memory allocation and GEMM.

PaRSEC-HiCMA-New with r the reallocation to the minimum size as needed during the factorization. The memory saved increases with the matrix size, up to 44× for this setting.

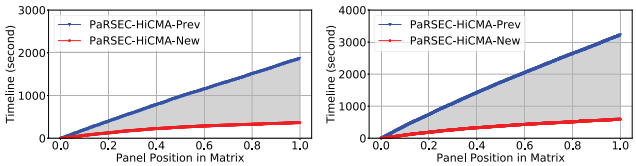
The right figure simulates the effect of memory reallocation in GEMM with b = 4500. It compares the time of a TLR GEMM with the cost of a memory allocation for the amount of 2×k×b (left y-axis) and the corresponding ratio (right y-axis) with variant k ∈ [13, 1079] (the actual minrank and maxrank of off-band tiles from this experiment on 512 nodes). The time for memory allocation is consistently more than two orders of magnitude cheaper than a TLR GEMM, and only TLR GEMMs with rank growing need to reallocate memory.

E. Comparison with State-of-the-Art

Existing alternative approaches based on low-rank approximations consist of standalone software solutions but rely all on the traditional, static, and rigid 2DBCDD for matrix computations on distributed-memory systems. They do not have mechanisms for handling load imbalance and dynamic memory allocation. Therefore, we only compare the performance of PaRSEC-HiCMA-New against state-of-the-art PaRSEC-HiCMA-Prev for the **st-3D-exp** application. Due to the different memory allocation strategies in the two libraries, we compared up to the largest problem size that could be executed with PaRSEC-HiCMA-Prev on 512 nodes with 128 GB of memory per node. For instance, PaRSEC-HiCMA-Prev could factorize matrix sizes up to 3.24M on 512 nodes (see Fig. 8) because of the memory limit per node 128 GB. Table II lists the performance traits for these matrix sizes. "Band-dense" shows the effect of BAND-DENSE-TLR algorithm and "hybrid distribution" with only (1)-POTRF recursive as that in

TABLE II: Performance comparisons.

No. of Nodes	Matrix Size	PaRSEC -HiCMA-prev (seconds)	Band-dense (seconds)	Recursive Kernels (seconds)	Total Speedup
64	1080000	1933.53	446.51	368.44	5.24X
128	1080000	1579.57	361.57	236.51	6.68X
256	1080000	1526.75	352.81	210.25	7.26X
512	1080000	1477.96	316.13	195.74	7.55X
256	2160000	3221.70	774.97	614.44	5.24X
512	2160000	3289.91	632.48	520.05	6.32X
512	3240000	5868.97	1536.10	1009.31	5.81X



(a) N = 1.08M on 64 nodes. (b) N = 2.16M on 256 nodes.

Fig. 9: Panel release time; x-axis is the panel position in matrix which is $panel_ID/$NT.

PaRSEC-HiCMA-Prev, and "recursive kernels" describes the impact of the recursive support for all numerical kernels. The major performance improvement comes from "Band-dense" because of (1) FLOPs reduction, as shown in Fig. 6(b), about 1.5×; (2) more balanced work-flows due to "hybrid distribution"; (3) improved parallelism exposed to runtime system due to smaller tile size BAND-DENSE-TLR algorithm can support(see Section VII-B). "Recursive kernels" further improves performance by shortening the critical path, which improves concurrency and expedites discovery of panel release. To highlight this part, Fig. 9 indicates the relative release time of each panel factorization for the two experiments in Table II. Each panel is released significantly earlier in PaRSEC-HiCMA-New than PaRSEC-HiCMA-Prev mostly because of the recursive dense GEMMs with more balanced work-flow instead of expensive TLR GEMMs close to the band which delay the panel release with accumulative effect.

F. Performance Evaluation at Scale

We first highlight how close the current performance is from the performance of the critical path as described above. Fig. 10 compares the time to solution on 512 nodes for different matrix sizes of the entire Cholesky factorization (All_kernels) compared with only the cost of the factorization on the dense part plus the panel (or the entire Cholesky factorization except for all low rank updates, No_TLR_GEMM), which is equivalent to the critical path at distance BAND_SIZE. The red lines indicate time to solution while the blue lines mean FLOPs. Although only a tiny fraction of FLOPs, No_TLR_GEMM contributes to most time-to-solution, because closer to the diagonal, and thus closer to the critical path, there is less available parallelism at each step. The time-to-solution ratio drops as matrix size increases, because (1) BAND_SIZE,

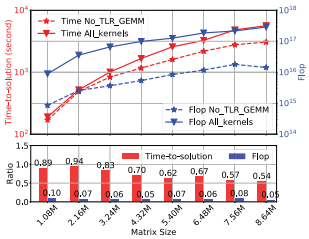


Fig. 10: Performance evaluation on 512 nodes.

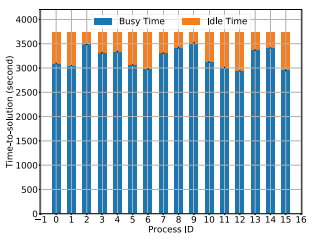


Fig. 11: Performance evaluation on 16 nodes.

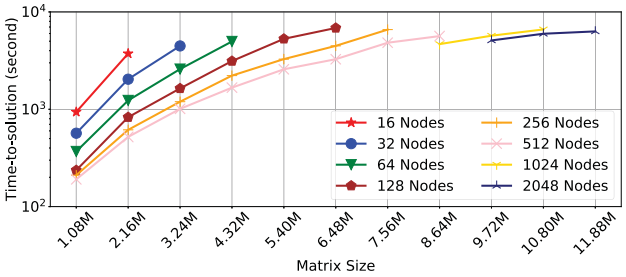


Fig. 12: Performance on Shaheen II.

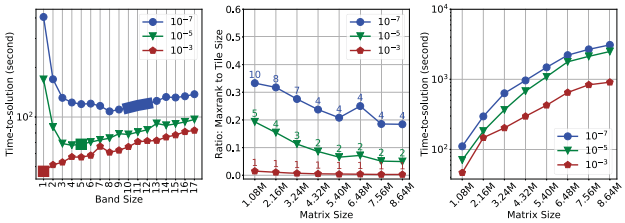


Fig. 13: Evaluation of accuracy thresholds on 512 nodes.

which is a tiny fraction of NT, decreases inversely proportional with the matrix size, as highlighted in Fig. 6(d); (2) the number of tiles on-band is $O(\mathtt{NT})$ while $O(\mathtt{NT}^2)$ tiles off-band exist.

Then, we measure the system usage by displaying the CPU's busy time and idle time of each process for N = 2.16M on 16 nodes (processes), as shown in Fig. 11. Load imbalance may happen among single-process nodes due to the static 2DBCDD, the irregular rank distribution and the rank variations (see Fig. 1) for off-band tiles. However, there is little imbalance among threads in a process, roughly achieving more than 90% CPU occupancy on average. In addition, the performance achieves 4.88 Tflop/s, which is about $1/3$ of the sustained Linpack performance on 16 nodes Shaheen II, i.e., 14.32 Tflop/s. TLR Cholesky is not purely compute-bound, since most flops comes from TLR GEMMs (see Fig. 10). TLR GEMM attains about one-third of the performance of a regular dense GEMM on a single core (see Fig. 2(a)).

Finally, Fig. 12 describes a large-scale performance evaluation with up to 2048 nodes and matrix size 11.88M ×11.88M The performance of each matrix size shows the strong scalability, and the graph for each node depicts weak scalability. The strong scaling improves as the matrix size increases, thanks to the high degree of parallelism. It is worth noting that we are still far away from the hardware memory capacity. For instance, the memory footprint needed per node for the maximal matrix size on 512 nodes (i.e., 8.64M) is 9.31 GB before factorization and 12.33 GB after factorization (see Fig. 8(a)), which is still far from the 128 GB memory capacity on the system. The dynamic memory designation may enable solving even large problem sizes for the same node budget.

G. Evaluation of Different Accuracy Thresholds

Time to solution is not the only metric of interest for TLR factorizations; it is also crucial to be able to provide the accuracy expected by the target science domain. In Fig. 13, we evaluate our algorithm from the standpoint of the accuracy threshold, its impact on the band size, and the time to solution. First, it is important to notice that the accuracy has a direct impact on the initial rank of the compressed tiles—a lower accuracy provides a faster decay of the ranks in the subdiagonals. These 3 additional accuracy threshold (10^{-7}, 10^{-5} and 10^{-3}) are complementary to the analysis above for 10^{-9}.

- Fig. 13(a) analyzes BAND_SIZE auto-tuning for three accuracy thresholds alike that for accuracy 10^{-9} shown in Section VIII-B. The rectangular boxes are the same range $fluctuation \in [0.67, 1]$ as before, and there is only one point within this range for accuracies 10^{-5} and 10^{-3}.
- The ratio_maxrank in Fig. 13(b) shows a rapid descent with the increase in matrix size, and with the decrease in accuracy. The autotuned band size tends to vary little and stabilize quickly. BAND_SIZE = 1 is always selected for accuracy 10^{-3} because of the large rank discrepancy for tiles on- and off- diagonal, similar to 2D applications.
- Finally, the time to solution depicted in Fig. 13(c) is consistent with the initial k and the expected FLOPs.

All in all, these results show the efficiency and scalability of PaRSEC-HiCMA-New for TLR Cholesky factorization, as well as its capability of delivering faster the results with the expected accuracy.

IX. Conclusion and Future Work

This paper demonstrates how a synergistic approach between HiCMA and PaRSEC based on a separation of concerns can improve the productivity, performance, and scalability of the challenging 3D exponential matrix kernel in the context of environmental applications. By propagating the rank information to the PaRSEC runtime, proper rank-aware runtime decisions are made for dynamic data structure adaptation, memory footprint optimizations, and data distribution. Using recursive formulations on tasks belonging to the critical path, we further expose concurrency to PaRSEC in order to shorten the makespan. Our resulting high-performance BAND-DENSE-TLR Cholesky outperforms previous implementations of data-sparse Cholesky factorization by up to 7-fold on a large-scale distributed-memory system, while minimizing the memory footprint up to a factor of 44-fold. For future work, we would like to provide dynamic load balancing between nodes to further mitigate the idle time. A more generic approach to BAND-DENSE-TLR will be to change the data structure on a tile basis instead of a band basis to capture tiles with high ranks located far away from the diagonal. Moreover, we would like to accelerate the tasks on the critical path using GPU hardware accelerators and combine it with mixed-precision algorithms.

Acknowledgments. This research was partially supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Access to computational resources, Shaheen II, has been generously provided by KAUST.

REFERENCES

[1] H. Meuer, E. Strohmaier, J. Dongarra, and H. Simon", "The Top500 List," June 2020, http://www.top500.org.

[2] D. E. Keyes, H. Ltaief, and G. Turkiyyah, "Hierarchical Algorithms on Hierarchical Architectures," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2020, pp. 1–11.

[3] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov, "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," in *Journal of Physics: Conference Series*, vol. 180, no. 1, 2009, p. 012037.

[4] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Herault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, and J. Dongarra, "Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*, May 2011, pp. 1432–1441.

[5] "The chameleon project," https://gitlab.inria.fr/solverstack/chameleon, January 2017.

[6] L. Grasedyck, D. Kressner, and C. Tobler, "A literature survey of low-rank tensor approximation techniques," *GAMM-Mitteilungen*, vol. 36, no. 1, pp. 53–78, 2013.

[7] M. S. Handcock and M. L. Stein, "A Bayesian Analysis of Kriging," *Technometrics*, vol. 35, pp. 403–410, 1993.

[8] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Parallel Approximation of the Maximum Likelihood Estimation for the Prediction of Large-Scale Geostatistics Simulations," in *2018 IEEE International Conference on Cluster Computing (CLUSTER)*, 2018.

[9] S. Börm, L. Grasedyck, and W. Hackbusch, "Introduction to hierarchical matrices with applications," *Engineering Analysis with Boundary Elements*, vol. 27, no. 5, pp. 405 – 422, 2003.

[10] S. Goreinov, E. Tyrtyshnikov, and A. Y. Yeremin, "Matrix-free iterative solution strategies for large dense linear systems," *Numerical Linear Algebra with Applications*, vol. 4, no. 4, pp. 273–294, 1997.

[11] W. Hackbusch, "A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices," *Computing*, vol. 62, pp. 89–108, 1999.

[12] R. Kriemann, "H-LU factorization on many-core systems," *Comput. Vis. Sci.*, vol. 16, no. 3, p. 105–117, Jun. 2013.

[13] E. Corona, P.-G. Martinsson, and D. Zorin, "An $O(N)$ direct solver for integral equations on the plane," *Applied and Computational Harmonic Analysis*, vol. 38, no. 2, pp. 284–317, 2015.

[14] F.-H. Rouet, X. S. Li, P. Ghysels, and A. Napov, "A distributed-memory package for dense hierarchically semi-separable matrix computations using randomization," *ACM Transactions on Mathematical Software (TOMS)*, vol. 42, no. 4, p. 27, 2016.

[15] S. Ambikasaran, D. Foreman-Mackey, L. Greengard, D. W. Hogg, and M. O'Neil, "Fast direct methods for Gaussian processes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 252–265, 2015.

[16] S. Börm, *Efficient numerical methods for non-local operators: H2-matrix compression, algorithms and analysis*. European Mathematical Society, 2010, vol. 14.

[17] T. Mary, "Block Low-Rank Multifrontal Solvers: Complexity, Performance, and Scalability," Ph.D. dissertation, Paul Sabatier University, Toulouse, France, November 2017.

[18] W. Hackbusch, "Survey on the Technique of Hierarchical Matrices," *Vietnam Journal of Mathematics*, vol. 4, p. 71–101, 2016.

[19] W. H. Boukaram, G. Turkiyyah, H. Ltaief, and D. E. Keyes, "Batched QR and SVD Algorithms on GPUs with Applications in Hierarchical Matrix Compression," *Parallel Computing*, vol. 74, p. 19–33, 2018.

[20] A. Charara, D. Keyes, and H. Ltaief, "Tile Low-Rank GEMM Using Batched Operations on GPUs," in *Euro-Par 2018: Parallel Processing*, M. Aldinucci, L. Padovani, and M. Torquati, Eds. Springer, 2018.

[21] K. Akbudak, H. Ltaief, A. Mikhalev, A. Charara, A. Esposito, and D. Keyes, "Exploiting data sparsity for large-scale matrix computations," in *European Conference on Parallel Processing*. Springer, 2018.

[22] Q. Cao, Y. Pei, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, "Extreme-scale Task-based Cholesky Factorization Toward Climate and Weather Prediction Applications," in *Proceedings of the Platform for Advanced Scientific Computing Conference*, 2020, pp. 1–11.

[23] Q. Cao, Y. Pei, T. Herault, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, "Performance Analysis of Tile Low-Rank Cholesky Factorization Using PaRSEC Instrumentation Tools," in *2019 IEEE/ACM International Workshop on Programming and Performance Visualization Tools (ProTools) at SC19*. IEEE, 2019.

[24] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. A. Mary, "Bridging the Gap Between Flat and Hierarchical Low-Rank Matrix Formats: The Multilevel Block Low-Rank Format," *SIAM Journal on Scientific Computing*, vol. 41, no. 3, pp. A1414–A1442, 2019.

[25] M. G. Genton, "Classes of kernels for machine learning: a statistics perspective," *J. Mach. Learn. Res.*, vol. 2, p. 299–312, Mar. 2002.

[26] H. He and W. Siu, "Single image super-resolution using gaussian process regression," in *CVPR 2011*, 2011, pp. 449–456.

[27] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J. Dongarra, "PaRSEC: A Programming Paradigm Exploiting Heterogeneity for Enhancing Scalability," *Computing in Science and Engineering*, vol. 99, p. 1, 2013.

[28] A. Danalis, G. Bosilca, A. Bouteiller, T. Herault, and J. Dongarra, "PTG: An Abstraction for Unhindered Parallelism," in *2014 Fourth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing*, 2014, pp. 21–30.

[29] R. Hoque, T. Herault, G. Bosilca, and J. Dongarra, "Dynamic Task Discovery in PaRSEC: A Data-flow Task-based Runtime," in *Proceedings of the 8th Workshop on Latest Advances in Scalable Algorithms for Large-Scale Systems*, ser. ScalA '17. ACM, 2017.

[30] A. Denis, E. Jeannot, P. Swartvagher, and S. Thibault, "Using dynamic broadcasts to improve task-based runtime performances," in *European Conference on Parallel Processing*. Springer, 2020, pp. 443–457.

[31] G. Morton, *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company, New York, 1966.

[32] K. Akbudak, H. Ltaief, A. Mikhalev, and D. Keyes, "Tile Low Rank Cholesky Factorization for Climate/Weather Modeling Applications on Manycore Architectures," in *32nd International Conference on High Performance, Frankfurt, Germany*. Springer International Publishing, 2017, pp. 22–40.

[33] E. Slaughter, W. Wu, Y. Fu, L. Brandenburg, N. Garcia, W. Kautz, E. Marx, K. S. Morris, W. Lee, Q. Cao, G. Bosilca, S. Mirchandaney, S. Treichler, P. S. McCormick, and A. Aiken, "Task bench: A parameterized benchmark for evaluating parallel runtime performance," *arXiv preprint arXiv:1908.05790*, 2019.