Extreme-Scale Task-Based Cholesky Factorization Toward Climate and Weather Prediction Applications

Qinglei Cao
qcao3@vols.utk.edu
University of Tennessee

Yu Pei
ypei2@vols.utk.edu
University of Tennessee

Kadir Akbudak
kadir.akbudak@kaust.edu.sa
King Abdullah University of Science and Technology

Aleksandr Mikhalev
aleksandr.mikhalev@kaust.edu.sa
King Abdullah University of Science and Technology

George Bosilca
bosilca@icl.utk.edu
University of Tennessee

Hatem Ltaief
hatem.ltaief@kaust.edu.sa
King Abdullah University of Science and Technology

David Keyes
david.keyes@kaust.edu.sa
King Abdullah University of Science and Technology

Jack Dongarra
dongarra@icl.utk.edu
University of Tennessee, the Oak Ridge National Laboratory and the University of Manchester, UK

Abstract

Climate and weather can be predicted statistically via geospatial Maximum Likelihood Estimates (MLE), as an alternative to running large ensembles of forward models. The MLE-based iterative optimization procedure requires the solving of large-scale linear systems that performs a Cholesky factorization on a symmetric positive-definite covariance matrix—a demanding dense factorization in terms of memory footprint and computation. We propose a novel solution to this problem: at the mathematical level, we reduce the computational requirement by exploiting the data sparsity structure of the matrix off-diagonal tiles by means of low-rank approximations; and, at the programming-paradigm level, we integrate PaRSEC, a dynamic, task-based runtime to reach unparalleled levels of efficiency for solving extreme-scale linear algebra matrix operations. The resulting solution leverages fine-grained computations to facilitate asynchronous execution while providing a flexible data distribution to mitigate load imbalance. Performance results are reported using 3D synthetic datasets up to $42M$ geospatial locations on $130,000$ cores, which represent a cornerstone toward fast and accurate predictions of environmental applications.

CCS Concepts

• **Mathematics of computing**; • **Computing methodologies** → **Massively parallel algorithms**;

PASC '20, June 29-July 1, 2020, Geneva, Switzerland

ACM ISBN 978-1-4503-7993-9/20/06…$15.00
https://doi.org/10.1145/3394277.3401846

Keywords

Low-rank matrix computations, Dynamic runtime system, Asynchronous execution, Load balancing, High performance computing

ACM Reference Format:
Qinglei Cao, Yu Pei, Kadir Akbudak, Aleksandr Mikhalev, George Bosilca, Hatem Ltaief, David Keyes, and Jack Dongarra. 2020. Extreme-Scale Task-Based Cholesky Factorization Toward Climate and Weather Prediction Applications . In *Proceedings of the Platform for Advanced Scientific Computing Conference (PASC '20), June 29-July 1, 2020, Geneva, Switzerland.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3394277.3401846

1 Introduction

Massive parallelism is the dominant force behind the increased capabilities of scientific computing. Given the structural constraints on technology and hardware design, high-performance computing (HPC) architecture development, which is striving to satisfy application needs and achieve new levels of performance, has to deal with unprecedented increases in concurrency, non-uniform hardware designs, and changing performance capabilities. In this unfriendly landscape, application developers face unfamiliar challenges at all levels, from the increase in the number of nodes to the highly complex architectural capabilities on each node, and from the lack of portability between different architectures to a lack of compatibility across different versions of the same hardware. Faced with such daunting challenges, combining existing programming paradigms (i.e., the so-called "MPI+X model") often backfires. The application programmer is exposed to the complexity of handling the non-uniform system explicitly, while the composition of multiple programming paradigms encourages a static distribution of the computation between different logical domains. As the systems grow increasingly complex, static assumptions about synchrony, deterministic scheduling, and predictable runtime of computation and communication alike, no longer bear out; and even a minor amount of system noise and small delays introduce significant slack in large-scale synchronous applications [16, 28, 50].

Consequently, it is becoming clear that to perform at extreme scales a shift in the programming model paradigm built around a far less synchronous approach is needed to help applications meet these challenges. The task-based programming model has proven to be both efficient and productive in this regard. In a task-based programming environment, a vast amount of parallelism is exposed through expressing the algorithm as a set of successive, fine-grain tasks (a set of instructions that access and modify an explicit and bounded amount of data). A runtime system is then responsible for scheduling these tasks while satisfying the data dependencies between them. Such a runtime must adapt to the changes in the amount of parallelism available in the application, and map that parallelism onto the underlying hardware resources under dynamic and hard-to-predict system conditions. Task-based programming models associated with dynamic runtime systems have been thoroughly studied and have demonstrated a leap forward in performance and programmability for many scientific computing fields—including application libraries built on top of the usual dense [4, 13, 19, 24, 27] and sparse [33, 35, 37, 48] linear algebra solvers with regular, arithmetic/memory-intense, computational tasks.

At the same time, covariance matrix problems have generated interest in the scientific community, thanks to the simplicity of their inherent symmetric matrix structures. In particular, they arise in models of choice for predicting climate and weather forecasting (i.e., environmental applications) [47], evaluating basis functions for electronic structure calculations (i.e., computational chemistry applications) [41], and identifying habitable galaxies (i.e., computational astronomy applications) [36], for which worldwide HPC supercomputing centers allocate a large number of their computing cycles. The size of these covariance matrices may significantly grow for very large input datasets and, therefore, make the arithmetic complexity and memory footprint unbearable.

The fundamental idea then is to exploit the low-rank or data sparsity structure by compressing the off-diagonal tiles of the dense covariance matrix up to a specific application-dependent accuracy. In particular, low-rank matrix computation, which represents a crucial class of matrix algorithms for geospatial statistics, can benefit from the same aforementioned task-based approaches or implementation. However, the properties of low-rank matrix computations make the integration more challenging because the granularity of tasks, in direct relationship to the rank of the data tile, varies due to the inherent heterogenous rank distribution, thereby raising the algorithmic load imbalance at the forefront. These elements present novel and supplementary burdens on runtime scheduling and motivate the expansion of dataflow runtime framework capabilities. This paper introduces the `Lorapo` library which demonstrates the impact of synergic opportunities between low-rank matrix computations and task-based runtime systems, i.e., `PaRSEC`, for solving the Maximum Likelihood Estimates (MLE) in the context of 3D climate and weather prediction applications. The MLE-based iterative optimization procedure requires the Cholesky factorization of large-scale, dense-symmetric, positive-definite covariance matrices, which is extremely demanding in terms of memory footprint and computation. Among other compression data formats, the `TLR` approach reduces the memory footprint and the computational requirement by exploiting the data sparsity structure of the matrix off-diagonal tiles by means of low-rank approximations. Numerical accuracy is ultimately preserved, just enough to maintain the statistical model fidelity for the prediction phase. The `PaRSEC` dynamic task-based runtime is then employed at the programming paradigm level to reach unparalleled levels of efficiency while performing extreme-scale linear algebra matrix operations toward solving environmental applications with up to $42M$ (Million) geospatial locations on $130,000$ cores. To the best of our knowledge, this work is the first to highlight performance of large-scale, task-based, and `TLR` Cholesky factorization for 3D scientific problems.

The remainder of this paper is as follows. Section 2 presents related work and Section 3 highlights the paper contributions. Section 4 describes the application and provides the necessary background for the `TLR` Cholesky factorization as well as the `PaRSEC` dynamic runtime system. Section 5 introduces the design and the novel implementations of the runtime optimizations to support `TLR` computational workloads. Performance results and analysis of the incremental optimizations are reported in Section 6. We conclude and present future work in Section 7.

2 Related Work

Numerous efforts are ongoing to support fine-grain dataflow programming. Recent task-based runtimes like `Legion` [14], `StarPU` [13], Open Community Runtime (`OCR`) [26], `OmpSs` [27], and `PaRSEC` [20] abstract the available resources to isolate application developers from the underlying hardware complexity and simplify the writing of massively parallel scientific applications.

`QUARK`, `OmpSs`, and `StarPU` provide a task insertion application programming interface (API) and dynamically build the task-graph. To interact with the runtime, the developer expresses sequential loop nests containing asynchronous task insertion calls. A consequence in distributed settings is that all of the participating processes have to discover the entirety of the graph to infer communication before reducing to the set of local tasks. This pruning phase limits potential scalability [32]. `QUARK` has no implicit support for heterogeneous nor distributed architectures though. `StarPU` provides automatic support for heterogeneous architectures and covers distributed execution via the insertion of implicit point-to-point communication tasks [3], which may prevent the benefits of potential collective communication. `OmpSs` follows a master-slave model where it allows nesting of tasks in individual nodes to relieve the master; however the master-slave model may suffer from scalability issues on distributed systems.

Recent versions of the `OpenMP` specification [40] introduce the *task* and *depend* clauses which can be employed to express dataflow graphs. `OpenMP` is widely used and supports homogeneous, shared-memory systems, and its *target* extension to support accelerators is quickly gaining traction. A limitation of the `OpenMP` model is that distributed-memory and internode communication needs to be described explicitly and performed with the use of an external communication library (e.g., MPI, SHMEM). `OCR`, still in early development stages, only supports homogeneous architectures. `Legion` describes logical regions of data, uses those regions to express the dataflow and dependencies between tasks, and defers the scheduling of tasks and data movement across distributed nodes to its underlying runtime, `REALM` [49].

On the applications side, climate and weather prediction applications that use geospatial statistics with MLE are prohibitively expensive due to high arithmetic complexity and large memory footprint. These applications necessitate direct matrix factorizations with $O(N^3)$ operations on $O(N^2)$ data, where N is the problem size, to directly compute the log-determinant and the linear solve involved in the MLE. This challenge prevents computational statisticians from increasing the scale or the details at which these problems need to be studied. Low-rank matrix operations may overcome this curse of dimensionality by using accuracy-tuned approximate methods. The mathematical theory behind low-rank matrix computations has been around for more than two decades [51]. Recent theoretical advancements have drastically improved the upper bounds for algorithmic complexity [11, 15].

In this context, there are several approaches to tackle the MLE, for instance, by exploiting the assumption of independence between blocks in the covariance matrix [46], by relying on Kronecker and Toeplitz algebra [52], and by using kernel-independent method [29]. All these aforementioned papers show performance results on 2D problems, up to 80K matrix size, and on single shared-memory node. While the former makes a strong assumption which may represent unrealistic situations in some cases, the two remaining approaches can be further accelerated with Hierarchically Semi-Separable (HSS) [54] and Hierarchical Off-Diagonal Low-Rank (HODLR) compression data formats [7], respectively. However, when solving 2D and 3D problems, these compression formats may eventually be subject to a substantial increase of their original arithmetic complexities, due to large off-diagonal ranks.

There are also methods which directly compute the matrix factorization to service the MLE. Owing to their hierarchy and recursive formulations, the first high-performance hierarchically low-rank LU factorization ($\mathcal{H}$-LU) implementation on homogeneous, shared-memory systems emerged only a few years ago [34], when more suitable programming models to support recursion (e.g., Cilk [18] and Intel Threading Building Blocks [44]) became available. Although these language features enhanced the user productivity, they may actually impede parallel performance because of the low hardware occupancy achieved on massively parallel systems.

Recent implementations of hierarchical low-rank matrix computations have been developed using the flat single-program, multiple-data (SPMD)/MPI programming model. This inherent bulk-synchronous approach relies on the static 2D block cyclic data distribution (2DBCDD) descriptor and maps the work onto resources with a global ordering between algorithmic steps, which are separated by global synchronous communication. Based on HSS compression data format, `STRUMPACK` [45] may achieve log-linear arithmetic complexity for problems with weak admissibility (i.e., typically 1D/2D problem matrices characterized by off-diagonal blocks with rather small ranks and homogeneous distributions). For 3D covariance problems studied herein, `STRUMPACK` may face challenges in compressing with HSS, since ranks may grow significantly, thereby rendering computations intractable. In addition, the resulting large discrepancy in the rank distribution may increase the communication volume due to the static 2DBCDD descriptor inherited from `ScaLAPACK` [17]. Strong admissibility data compression formats, such as block low-rank (BLR) [8, 43] and $\mathcal{H}^2$-like fast multipole method (FMM) [55] may better support the compression of 3D problems with a log-linear complexity for the latter. However, as implemented in MUMPS [10], BLR may also suffer from over communicating and load imbalance due to the heterogeneous rank distributions and the incapacity of the static 2DBCDD descriptor to address it. This is perhaps the reason why the factorization of the dense fronts are currently only performed on shared-memory systems using the fork-join bulk synchronous paradigm from the OpenMP programming model [9]. Matrix-free methods like FMM [30] have demonstrated their effectiveness in performing task-based hierarchical compression of covariance-like matrices [55], but the class of matrix factorization algorithms remains an open research problem. Furthermore, previous work on task-based, `TLR` Cholesky factorization in the context of `HiCMA` library [1, 2, 5], in which local tasks are scheduled by `StarPU` task-based dynamic runtime, already highlighted performance bottlenecks for 2D problems. Indeed, due to the static 2DBCDD descriptor—as well as the lack of support for collective communications—the resulting implementation have shown a distressing lack of scalability.

All in all, these low-rank matrix operations deal with heterogeneous workloads, which result in load imbalance during computations and communications. The bulk synchronous programming model, along with the static 2DBCDD descriptor—on which the dense linear algebra community has been relying for more than two decades—lacks the necessary features to mitigate the load imbalance. These challenges are further exacerbated when solving 3D problems due to a larger rank discrepancy among blocks. Fine-grained computations with a flexible dynamic runtime system become paramount when supporting such workloads at scale. However, to fully address the load imbalance issue at the source, a careful consideration of matrix data sparsity patterns should be adopted. This requires decoupling the expression of the numerical algorithm itself from its data mapping onto the system memory. This separation of concerns turns out to be a key in scaling up low-rank matrix computations on massively parallel systems.

3 Contributions

The challenges we are addressing in this paper are two-folds, at the algorithmic and programming paradigm level, both arising from the variability introduced by the low-rank approximations. More precisely these challenges are: a need for irregular data distributions to address the imbalance in memory consumption and in computational load, dynamic communication needs that must adapt to the variable tile ranks, and a mismatch between the traditional load-balancing and lookahead techniques and the needs of `TLR` Cholesky. Prior efforts to implement `TLR` Cholesky have not addressed these challenges and as a result they suffers from hard limitations on the accessible problem size, a general lack of scalability with the increasing size of the execution environment and an inability to address domain science with a higher discrepancy in ranks resulting from 3D problems.

The following innovations represent the core of our efforts and the driving story of this paper:

(1) deploying the `PaRSEC` task-based runtime with its inherent features (e.g., hybrid/flexible descriptor for data distribution)

to mitigate the 2DBCDD overheads, while synergistically tackling TLR matrix computations in the context of Lorapo;
(2) optimizing runtime execution via communication-reducing and synchronization-reducing techniques;
(3) simulating matrix covariance kernels as proxy for environmental applications based on geospatial statistics; and
(4) performing large-scale TLR Cholesky factorization up to $42M$ geospatial locations on a distributed-memory system with up to 130,000 cores.

To the best of our knowledge, this is the first time a dense, TLR Cholesky factorization has been deployed at this scale with an unprecedented time-to-solution using large, 3D, synthetic datasets generated from covariance matrix kernels used as proxy for geospatial statistics. The TLR Cholesky factorization corresponds to the most time-consuming operations when calculating the Maximum Likelihood Estimates (MLE), and plays, therefore, a pivotal role toward solving climate and weather prediction applications.

4 Background

This section provides detailed information on the geospatial statistics model used for climate and weather prediction applications, recalls the TLR Cholesky factorization, and describes the PaRSEC dynamic runtime system used in the paper.

4.1 Climate and Weather Prediction Model

The Gaussian process is one of the state-of-the-art models used for climate and weather prediction applications. Physical properties like temperature, wind speed, or soil moisture are assumed to be random values following normal distributions with a given mean and deviation. The behavior of these properties are observed at various spatial locations, and the main idea is to use these observations and their corresponding Gaussian processes to predict missing field values. This prediction phase defines the core of *geospatial statistics*. The interactions between all pairwise spatial locations constitute the basis for building the covariance matrix for the considered property. This paper focuses on a representative covariance matrix kernel: a square exponential function, usually called a "Gaussian radial basis function:"

$$f(x,y) = e^{-\frac{r^2(x,y)}{2l^2}}, \tag{1}$$

where $r(x,y)$ is the Euclidian distance from x to y, and $l > 0$ is the covariance length. The size of the resulting dense covariance matrix is as large as the number of spatial locations, which can be on the order of billions. If the covariance matrix can be generated, statistical parameters have to be computed through the MLE:

$$L(\theta) = -\frac{1}{2} z^T \Sigma^{-1}(\theta) z - \frac{1}{2} \log |\Sigma(\theta)|, \tag{2}$$

where θ corresponds to all covariance statistical parameters, z is the actual vector of observations (e.g., temperature), and $\Sigma(\theta)$ is the covariance matrix itself. Covariance matrices, based on Gaussian radial basis function, are symmetric and positive definite for any values of θ, as long as all spatial points are distinct. In this paper, we limit the list of parameters to a single one (i.e., the covariance length l), and we set it to $l = 0.1$ since it is used to represent a medium relation, with all the spatial points belonging to unit square (2D) or unit cube (3D). Parameter optimizations are beyond the scope of this paper and have been extensively studied [1]. In Equation (2), the symmetric, positive-definite, covariance matrix is used in two operations: (1) the linear solver and (2) the calculation of the determinant. In this paper, we employ the Cholesky factorization of the covariance matrix for both of these matrix operations: the former needs the Cholesky factor for the forward and backward substitutions, and the latter corresponds to the product of the diagonal entries of the Cholesky factor. Unfortunately, for a large number of geospatial locations, the dense Cholesky factorization is intractable due to the cubical algorithmic complexity, while the memory footprint incommensurate with current systems. Low-rank matrix approximation and computation become utterly critical to alleviate both aforementioned bottlenecks.

4.2 Tile Low-Rank Cholesky Factorization

To better understand the TLR Cholesky factorization, we first briefly revisit the classical tile algorithms for dense linear algebra [4]. The matrix is first decomposed into dense tiles. The standard dense Cholesky factorization usually operates on the underlying tile data layout by subsequently calling the four computational kernels *POTRF* (Cholesky factorization), *TRSM* (triangular solve), *SYRK* (symmetric rank k update), and *GEMM* (general matrix multiply) on the lower or upper part of the symmetric matrix. The whole factorization translates into a directed acyclic graph (DAG), where nodes correspond to tasks, and edges represent data dependencies. The critical path of the DAG, which is the serial and incompressible path, is $(NT-1)\times(POTRF+TRSM+SYRK)+POTRF$, where NT is the number of row/column tiles, and the other four variables are the execution time of the respective kernels. Unfortunately, main memory becomes the limiting factor when dealing with large matrix sizes for dense problems.

TLR approximations come to the rescue to address the curse of dimensionality by exploiting the data sparsity structure of the matrix operator. Reordering of rows/columns may be necessary to further expose the low-rankness of the off-diagonal tiles, which can then be approximated up to the application-dependent accuracy threshold by using a variant of the singular value decomposition (SVD), e.g., based on QR/divide-and-conquer algorithms [12] or even a faster approach based on randomized techniques [31]. This is the case for the square exponential covariance function studied herein in Equation (1), thanks to its asymptotic smoothness [22, 51]: the decay of singular values of the covariance matrix of any two sets

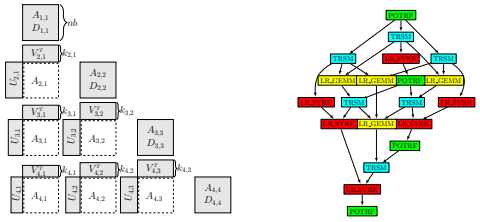


Figure 1: On the left, TLR format for matrix A having 4-by-4 tiles of size nb-by-nb. Diagonal tiles, $D_{i,i}$, are stored as dense. Off-diagonal tiles, $A_{i,j} = U_{i,j}V_{i,j}$, are compressed. Each off-diagonal tile, $A_{i,j}$, has its own rank, $k_{i,j}$. On the right, the corresponding DAG for Lorapo_POTRF on the same matrix.

Algorithm 1 Lorapo_POTRF(D, U, V, NT, acc)

```
for p = 1 to NT do
    POTRF(D(p,p))
    for i = p+1 to NT do
        TRSM(V(i,p), D(p,p))
    for j = p+1 to NT do
        LR_SYRK(D(j,j), U(j,p), V(j,p))
        for i = j+1 to NT do
            LR_GEMM(U(i,p), V(i,p), U(j,p), V(j,p), U(i,j), V(i,j), acc)
```

Table 1: Arithmetic Complexity: Dense vs. TLR Cholesky.

Kernel	Dense Cholesky	TLR Cholesky
POTRF	$\frac{1}{3} \times nb^3$	$\frac{1}{3} \times nb^3$
TRSM	nb^3	$nb^2 \times k$
SYRK/LR_SYRK	nb^3	$2 \times nb^2 \times k + 4 \times nb \times k^2$
GEMM/LR_GEMM	$2 \times nb^3$	$36 \times nb \times k^2$
Total	$O(N^3)$	$O(N^2 k)$

of spatial locations depends only on the relation of distance between these sets to the maximum of their diameters (a.k.a. admissibility condition). To improve the ratio of distance to diameter and reduce ranks, we have to cluster the spatial locations, corresponding to consecutive rows or columns. For this purpose, we employ the Morton ordering scheme [39], also known as the Z-order scheme. This is not the only possible ordering method, Peano curve [42] being another famous plane-filling curve.

The resulting TLR compression data format is shown in Figure 1 for a 4-by-4 matrix of tile size *nb*-by-*nb*. The diagonal tiles remain dense, since the pairwise correlations may be stronger (rank $k = nb$) than the cross correlations represented in the low-rank, off-diagonal tiles (with $k << nb$ for tiles farther away from the diagonal tiles).

TLR is the *de facto* compression data format for the HiCMA library [2, 5, 6]. To work on the compressed data layout of the off-diagonal tiles, the HiCMA library mainly necessitates developments of new low-rank *LR_SYRK* and *LR_GEMM* kernels, which requires decompression and recompression phases respectively, as introduced in [6]. To make the paper self-contained, we recall pseudo-code of the sequential TLR Cholesky algorithm in Algorithm 1.

Although the data layouts are different between dense and TLR Cholesky factorization, the DAG remains the same and so does the length of the critical path. Let us assume L is the number of tasks along the critical path, and D is all of the other operations except L; this set of operations is almost embarrassingly parallel. Therefore, the execution time will be the maximum between L and D/C, with C being the number of computational resources. It should be noted that, for the critical path, we ignore all cost related to data movements as if this critical path executes on a single node; and for the parallel part, we disregard all data dependencies as if all kernels could be executed in parallel and, therefore, be perfectly scalable with the number of computational resources.

Looking at the arithmetic complexity of individual tasks in Table 1, we can derive the minimal operations count $O(N^2 k)$ attained when $nb = O(\sqrt{N})$. The detailed complexity analysis of TLR Cholesky factorization can be found in [38]. It is also worth noting both the three Level-3 BLAS kernels and their arithmetic complexities are redefined for TLR matrix computations (introduced in [6]), creating severe situations of load imbalance for TLR Cholesky factorization, which do not exist in the dense Cholesky factorization.

4.3 The PaRSEC Runtime System

As a task-based runtime for distributed heterogeneous architectures, PaRSEC [20] is capable of dynamically unfolding a description of a graph of tasks on a set of resources and satisfying all data dependencies by efficiently shepherding data between memory spaces (between nodes but also between different memories on different devices) and scheduling tasks across heterogeneous resources. The overall PaRSEC programming model focuses on overcoming four main barriers to algorithm scalability and efficiency:

(1) *starvation*, insufficient concurrent work available to maintain high utilization of all resources;
(2) *latency*, the time-distance delay intrinsic to accessing remote resources and services and delays due to oversubscribed shared resources;
(3) *overhead*, the work required for the management of parallel actions and resources on the critical path of execution, which is not necessary in a sequential variant; and
(4) *heterogeneity*, support for specialized hardware to maximize performance (accelerators) and minimize overheads (smart communication hardware/NIC).

PaRSEC facilitates design of domain specific languages (DSL) [21] that allow domain experts to focus on their science rather than on the computer science. These DSLs rely on a dataflow model to create dependencies between tasks and target the expression of maximal parallelism.

The Parameterized Task Graph (PTG) [25] DSL uses a concise, parameterized, task-graph description known as Job Data Flow (JDF) to represent the dependencies between tasks. To enhance the productivity of the application developers, PaRSEC implicitly infers all communications from the expression of the tasks, supporting one-to-many and many-to-many types of communications. From a performance standpoint, algorithms described in PTG have been shown capable of delivering a significant percentage of the hardware peak performance on many hybrid distributed machines, as highlighted in [25], where for instance DPLASMA, a dense linear algebra (DLA) library using PaRSEC, yields superior performance compared with the most widely used DLA library, ScaLAPACK [17] or compared with state-of-the-art computational chemistry applications.Other DSLs, such as Dynamic Task Discovery (DTD) [32], are less science-domain oriented and provide alternative programming models to satisfy more generic needs by delivering an API that allows for sequential task insertion into the runtime. This programming model is simple and straightforward and has been shown to deliver good performance on small and medium-sized platforms. However, it suffers from the sequential discovery of tasks that hinder its scalability, similar to StarPU and QUARK. Indeed, as it stands from Algorithm 1, the TLR Cholesky may scale up to a limited number of nodes if DTD programming model is employed (see again [5]). The central idea is to empower Algorithm 1 with the PTG as the driving engine for performance scalability.

5 Optimizations for TLR Algorithms

We optimize runtime performance based on three criteria: improving load balancing, limiting memory usage and shortening the execution time of the critical path. The first two are addressed using a carefully designed data distribution, while the last one takes

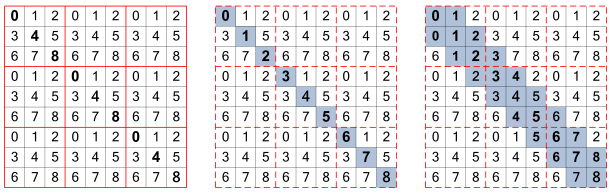
advantage of PaRSEC features conveniently exposed via the PTG DSL that allows us to drive the execution of the algorithm following a critical path. With the flexibility provided by the PaRSEC runtime, we: (1) introduce a new hybrid data distribution to improve load balancing; (2) reduce communication volume to limit memory usage; (3) leverage a new lookahead scheme to enforce critical path execution more aggressively to reduce waiting time in the critical path; and (4) deploy dense, hierarchical *POTRF* (i.e., nested parallelism) to reduce the critical path execution time.

5.1 Hybrid Data Distributions

Imbalance arises in TLR algorithms from two sources, related to a single root cause: the rank disparities between tiles on and off diagonal. The first source of imbalance is memory as the memory needed to store compressed tiled is directly proportional to its ranks ($rank * nb$), while the dense, diagonal tiles require $nb * nb$. The second is the computational costs to apply operations on these denser tiles, since denser tiles have a higher computational cost ($O(nb^3)$) compared with compressed tiles ($O(nb^2k)$), as highlighted in Table 1. Thus, the tiles closer to the diagonal pose two threats: they require significant storage and impose a high computational burden compared with the rest. It is therefore critical to ensure a more even distribution of these dense tiles across all available computational resources. Such a data distribution is unfamiliar in today HPC world, where the highly regular 2D block cyclic data distribution (2DBCDD), or ScaLAPACK 2D block cyclic, rules.

It is worth mentioning that 2DBCDD has been proven the optimal data distribution for most dense linear algebra operations, including the dense Cholesky factorization. First, because all tiles being equal, both the memory and computational burden is well distributed across processes, and second because the simple mapping onto a 2D cartesian process grid leads to simpler code to describe the communication patterns in today's *de facto* programming paradigm, MPI. However, in our case low rank tiles destroy the balance of 2DBCDD and our supporting DSL can automatically infer communications, so we are free to explore more suitable data distribution patterns.

To mitigate the load imbalance of a 2DBCDD, we defined a new, slightly less regular, data distribution scheme called "band distribution". This "band distribution" allows us to more evenly distribute the diagonal tiles across all participating processes, while falling back to 2DBCDD for the remaining off-diagonal tiles. To the best



(a) **(b)** **(c)**

Figure 2: (a) 2D block cyclic data distribution on a 3×3 process grid; (b) Band distribution with size $b = 1$, and a 9×1 process grid over a 3×3 off-band process grid; (c) Band distribution with size $b = 2$, and a 9×1 process grid over a 3×3 off-band. Numbers represent process ID.

of our knowledge, PaRSEC is the first task-based runtime that can handle such hybrid data distribution to support data heterogeneous workloads, as seen in low-rank matrix approximations. One can visualize this data distribution as two intertwined 2DBCDD using different process grids and superposed together, as shown in Figure 2 (b) and (c). In such data distribution, the diagonal and all tiles up to a distance b from them will use a different process grid from the rest of the tiles. As an example in Figure 2 (c), the band with $b = 2$ uses a process grid of 9×1 while the rest of the matrix uses a process grid of 3×3. Using this new band distribution, the load imbalance challenge, both in memory and computations, can be overcome by arranging tiles within band size b in a round-robin fashion and tiles outside the band b in a normal 2DBCDD. The band distribution can handle load imbalance issues in TLR Cholesky along with other load imbalance issues which may be caused by disproportionate time complexity (e.g., emerging from mixed precision calculations). It must be mentioned that such hybrid data distributions for a single matrix object are not supported on any of the dense linear algebra libraries available today, and certainly not ScaLAPACK, the most widely available one.

5.2 Reduce Communication Volume

Since the computation intensity is much lower in TLR Cholesky than in its dense counterpart, inter-node communication is bound to play a critical role in the performance of the algorithm. Although we define a maximum rank for the off-diagonal tiles, for the majority of them, the actual rank during the execution is always lower than this maximum rank and might vary during the factorization. Using the constant maximum rank for all communications is enticing, as it facilitates the algorithm coding and provides a portable across runtimes and easy to implement solution. Unfortunately, this simple approach leads to an increase in the volume of communications, as we are bound to always transfer more data than needed ($maxrank * nb$ instead of $rank * nb$ per communication). This overhead increases with the distance to the diagonal, reaching for low ranks tiles orders of magnitude ($\frac{maxrank}{rank}$). We can approximate the total reduction in communication volume by the *maxrank* divided by the average rank across all non-diagonal tiles. Moreover, as the algorithm communication needs increase rapidly as the matrix grows, the maximum network bandwidth may be reached, and the communications will then become one of the critical bottlenecks, with a direct negative impact on the overall performance.

The PaRSEC runtime, used by the Lorapo library, provides mechanisms for sending variable sizes data to remote processes, even when this size is dynamically decided by the task producing the corresponding data. This feature is unique in the task-based runtime world, as most of the runtimes mentioned in related work are still trying to cope with mostly regular, dense cases. Taking advantage of this PaRSEC capability makes it possible to decrease data transfers to only the actual tile data rank, therefore reaching communication optimality. Such feature may alleviate the bandwidth saturation and communication overhead, while releasing memory pressure on the receiver side.

5.3 Lookahead to Emphasize the Critical Path

PaRSEC enables tasks as soon as all their dependencies are available, and can therefore enable maximum parallelization without

the constraint of sequential code visibility, or window size, for task insertion. This way, PaRSEC can maximize the number of potential ready tasks, while agilely confronting the scheduling burden to drive the execution in a way that minimizes the resource idleness and global synchronizations. Even if the scheduler takes into account data locality, global and local constraints, and allows work stealing within local computational resources, dealing with large number of unordered tasks may become either a performance bottleneck or a performance hazard. As explained in Section 4.2, task priority for TLR Cholesky becomes paramount to aggressively follow the critical path. The runtime may certainly also maximize the performance of the parallel part, but may restrict the use of these tasks as fill-in for the lack of parallelism in the critical path.

As a consequence, a delay in the critical path may have more than a local impact, since it can propagate to remote processes resulting on a significant disturbance, creating a cascading effect of increasing delays, and therefore, a lower hardware occupancy across the distributed resources. The PaRSEC concept of control dependency between tasks can be used to guide the task execution order and priorities, as its only purpose is to add empty dependencies to delay tasks readiness. Taking advantage of this control dependency, we extend the existing *POTRF* implementation by adding novel lookahead techniques, different from the traditional left- or right-looking in the classical dense Cholesky.

To prioritize tasks on the critical path, a control dependency between *LR_SYRK* and *TRSM* of the same panel factorization is used, delaying the discovery of parallelism outside the critical path (corresponding to the update operation). More precisely, this control dependency applies to some *TRSM* operations (few rows away from the current *POTRF*), and indirectly propagate to other operations (mainly *LR_GEMM*). For instance, the $TRSM(m,k)$ kernels (m represents the tile row index and k the panel factorization index) with the $m > lookahead + k$ (*lookahead* defines the number of *TRSM* operations delayed in each panel factorization) are delayed by $LR\_SYRK(k,m)$ $(m = k{+}1)$ until the corresponding $POTRF(k{+}1)$ is executed. By varying *lookahead* the critical path is unfolded at the right pace, ensuring the prioritization of the critical path and a higher hardware occupancy.

5.4 Hierarchical *POTRF*

Since the dense *POTRF* kernel on the diagonal dense tiles has a computational intensity of at least an order of magnitude larger than the other three Level-3 BLAS kernels (that are applied on low-rank tiles), in addition to being located in the critical path of the algorithm, we need to promote this kernel and execute it as fast as possible in order to shorten the critical path and reduce the *POTRF* execution time. Speeding up *POTRF* will also reduce the waiting time for other cores by minimizing the potential starvation—particularly at the end of the execution where the opportunities for parallelism are lesser. Previous work in PaRSEC [53] introduced the idea of hierarchical DAG scheduling for hybrid distributed systems, where the task granularity is dynamically adjusted to adapt the algorithm to the available computational resource on the node, and to match their computational capabilities. In this paper, we extended this idea to TLR Cholesky by hierarchically creating a node-local task pool that decomposes the *POTRF* kernel on diagonal dense tiles into smaller subtiles to expose nested parallelism, and ensures work is available for all computational resources. This approach has the potential to improve core utilization (thus occupancy) and reduce the cost of the critical path.

6 Performance Results and Analysis

The experiments are run on Shaheen II, a Cray XC40 system. which has 6,174 compute nodes, each with two 16-core Intel Haswell CPUs running at 2.30 GHz and 128 GB of DDR4 main memory. All calculations are performed in double-precision floating-point arithmetic. In all experiments, numerical backward errors have been consistently validated against the application accuracy threshold to ensure correctness. In particular, we compress off-diagonal tiles and retain their most significant singular values (and associated vectors) above the accuracy threshold of 10^{-8}, which ultimately yields absolute numerical error of order 10^{-9} in the solution of linear system in Equation (2). This 10^{-9} tolerance is sufficient to satisfy the prediction accuracy requirements of the 3D climate and weather prediction applications, as described in [2]. We employ a process grid $P \times Q$ across computational nodes and make it as square as possible, otherwise suitable P and Q, where $P < Q$. We run our experiments at least three times; since no major performance variability has been noticed throughout our experiments, so the minimum time to solution is reported. For the execution with the largest number of cores, we make a single run to save on core hours. It is worth noting that the most suitable tile size is used for all experiments based on Section E in [23].

6.1 Application Settings

The numerical experiments use synthetic and realistic covariance matrix kernels from three application settings:

(1) **syn-2D**: synthetic problem on a plane with the following covariance function: $f(x,y) = \frac{\sin(\lambda r(x,y))}{r(x,y)}$, which corresponds to the imaginary part of the fundamental solution of the Helmholtz equation and is usually called the *sinc* function. Although this function is not asymptotically smooth, it gives good TLR matrix approximations with more-or-less equal ranks of off-diagonal tiles. The parameter λ is a wave number corresponding to the number of wave oscillations per unit distance and is set to 100. We use this synthetic kernel only to demonstrate the robustness of the proposed software framework and for performance analysis.
(2) **st-2D-sqexp**: spatial statistics problem on a plane with a square exponential covariance function, as introduced in Equation (1) from Section 4.1 in the context of climate and weather prediction applications.
(3) **st-3D-sqexp**: spatial statistics problem in a 3D space with a square exponential covariance function, which the 3D extension of the Equation (1) from Section 4.1.

The spatial locations for each application are generated as follows. Given N, we find exact square (for problem on plane/2D space) or cube (for problem in 3D space) number M not less than N. We generate uniform distribution of M points in $unit^2$ or $unit^3$ domain. We then sort all M points by the first coordinate—all points with the same coordinate are sorted along the second axis, and then the same happens along the third axis (for 3D space). We pick the first

N points out of M sorted points. And finally, we apply the Z-order sorting scheme for N picked points.

The generation of a TLR matrix consists of two phases: (1) generation on-the-fly and (2) compression. For each off-diagonal tile $A_{i,j}$ of a matrix A, a temporary dense matrix is generated and compressed into $U_{i,j}$ and $V_{i,j}$ factors using randomized SVD [31], whereas the diagonals are kept in dense format. The ratio of the time to generate and compress over the factorization time decreases due to the difference in the asymptotic complexities of the two phases [5]. Thus, we only focus on reporting the time of the TLR Cholesky factorization in the subsequent sections.

Furthermore, as opposed to **syn-2D**, the two statistics applications show more discrepancy (more than 2.8× in final average and maximum ranks) in rank distribution. In other words, the ranks in **syn-2D** are observed to be more homogeneous with respect to the statistics applications. Among the statistics problems, the difference between average and maximum ranks is the smallest for **st-2D-sqexp** and the largest for **st-3D-sqexp**. *Higher discrepancy in ranks results in higher imbalance in computation and communication. Hence, sophisticated task and data distribution heuristics and a dynamic runtime become important to efficiently solve such problems.*

6.2 Comparison with HiCMA

We compare the performance of the proposed TLR Cholesky implemented with Lorapo against HiCMA using **syn-2D** (Figure 3a) and **st-2D-sqexp** (Figure 3b), the only two supported applications with results reported in [5] using HiCMA. It should be noted that the Lorapo version includes all optimizations noted in Section 5. In these two figures, results up to $11M$ are given in Lorapo to compare to HiCMA in compliance with Figure 8 of [5]. Some points are missing from the figures owing to memory limitations. The Lorapo implementation scales to much larger matrix sizes due to its better memory management, hybrid data distribution and reduced communication volume, as described in Section 5, allowing the factorization to be scaled up to a $32M$ matrix size on 512 nodes for **st-2D-sqexp** (Figure 6). Moreover, Lorapo consistently outperforms HiCMA. In fact, the performance of Lorapo on 64 nodes for both **syn-2D** and **st-2D-sqexp** is better than any HiCMA configurations' results. When the matrix size is small, increasing the number of nodes does not improve performance, because the time to solution is dominated by the sequential critical path, L. However, as we increase the matrix size (e.g., $10M$), the balance between L and D, and the performance improves, and a larger number of processors delivers better performance. On the HiCMA side, the performance declines further for **syn-2D**, and—as seen in Figure 3a—performance on 512 nodes is almost the worst, highlighting a lack of scalability in their implementation.

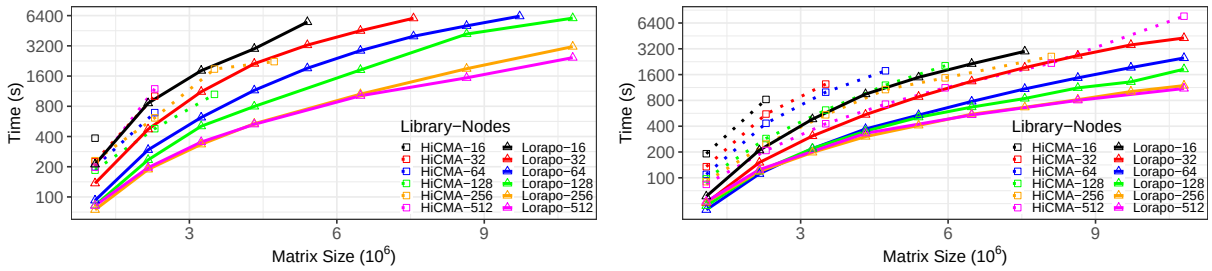
6.3 Effect of Proposed Runtime Optimizations

In this section, we analyze in detail the impact of each one of four optimizations described in Section 5 on one of the statistical applications, **st-3D-sqexp**. We are using the following abbreviations with regard to the four optimizations: **NONE** for **no optimizations**, **B** for **B**and distribution, **BS** for **B** and **S**ending actual rank during runtime, **BSL** for **BS** and **L**ookahead to enforce critical path execution, and **BSLH** for **BSL** and **H**ierarchical *POTRF*. All of these experiments are run on 16, 32, 64, 128, and 256 nodes. In Section 6.3.5 we summarize all optimizations into a single consistent graph.

6.3.1 Hybrid Data Distributions. Figure 4a shows the effect of the proposed load balancing technique for **st-3D-sqexp** when compared to no optimization (**NONE**). A band size $b = 1$ is used in band distribution to compensate for the imbalance occurring on diagonal tiles due to rank discrepancies between tiles on and off diagonals. Although diagonal tiles are on the critical path, fewer tasks are applied on them at each iteration: a single *POTRF* and a number of *LR_SYRK* (depending on the tile position in the matrix). However, since diagonal tiles are full rank, the tasks on the diagonal tiles become more compute intensive than the rest of updates (*TRSM* and *LR_GEMM*). In particular for the TLR Cholesky factorization, a *POTRF* and a large number of *LR_SYRK* (one per tile below the *POTRF* tile position) can be executed in parallel, leading to a surge in compute intensive tasks (because they apply on full dense tiles) that are all on the critical path. To alleviate the burden of these time-consuming tasks from the critical path, we rely on the band distribution to execute in parallel all these operations across the maximum number of resources. This differs from the traditional 2DBCDD in Figure 2a, for which diagonal tiles are only spread across a subset of diagonal processes in the process grid distribution. The data from the off-diagonal tiles, on which tasks outside of the critical path operate on, are still distributed using the traditional 2DBCDD. All in all, the resulting hybrid data distribution, i.e., band distribution combined with 2DBCDD, is utterly important to scale on massively parallel systems.

6.3.2 Reduce Communication Volume. PaRSEC can handle dynamically sized data, providing Lorapo with the opportunity to only send the necessary data ($rank * nb$ instead of $maxrank * nb$). This not only decrease the communication volume and thus overhead, but also significantly reduce memory usage on the receiver, because the receive buffer can now be tightly allocated with the real *rank* instead of *maxrank*. The volume reduction being data dependent (on *rank*) it is difficult to estimate it accurately. Considering the exact same case as above, 3.24M matrix size on 256 nodes as an example, **BS** reduced the data transfer volume by $\frac{400}{32.54} = 12.3\times$. As indicated in Figure 4b, the decrease in required memory on the receiver side allows for solving significantly larger problems (up to $10M$ instead of only $6M$), while providing the means to reduce the time to solution by about 25% using the same matrix size from the previous approach.

6.3.3 Lookahead to Emphasize the Critical Path. Figure 4c reveals the impact of the proposed lookahead technique, **BSL**, compared to **BS** for **st-3D-sqexp**. Smaller *lookahead*, more constraint added to runtime to limit the potential parallelism. In practice the higher discrepancies between tiles on and off diagonal, smaller number of resources and bigger the problem size, the smaller *lookahead* should be. Hence, several *lookahead*s are experimented and the best time-to-solution is reported. In this figure, we can see the benefit of lookahead grows with the matrix size, but decreases with the number of nodes. The reason behind is that the lookahead hints provided to the runtime and used to prioritize the critical path executions are more impactful when there is an abundance of work, so when the matrix increases over a fixed number of resource or the number of resources decreases for the same problem size.



(a) syn-2D. (b) st-2D-sqexp.

Figure 3: Performance comparison of the proposed TLR Cholesky framework with `Lorapo` and `HiCMA` for the 2D kernels.

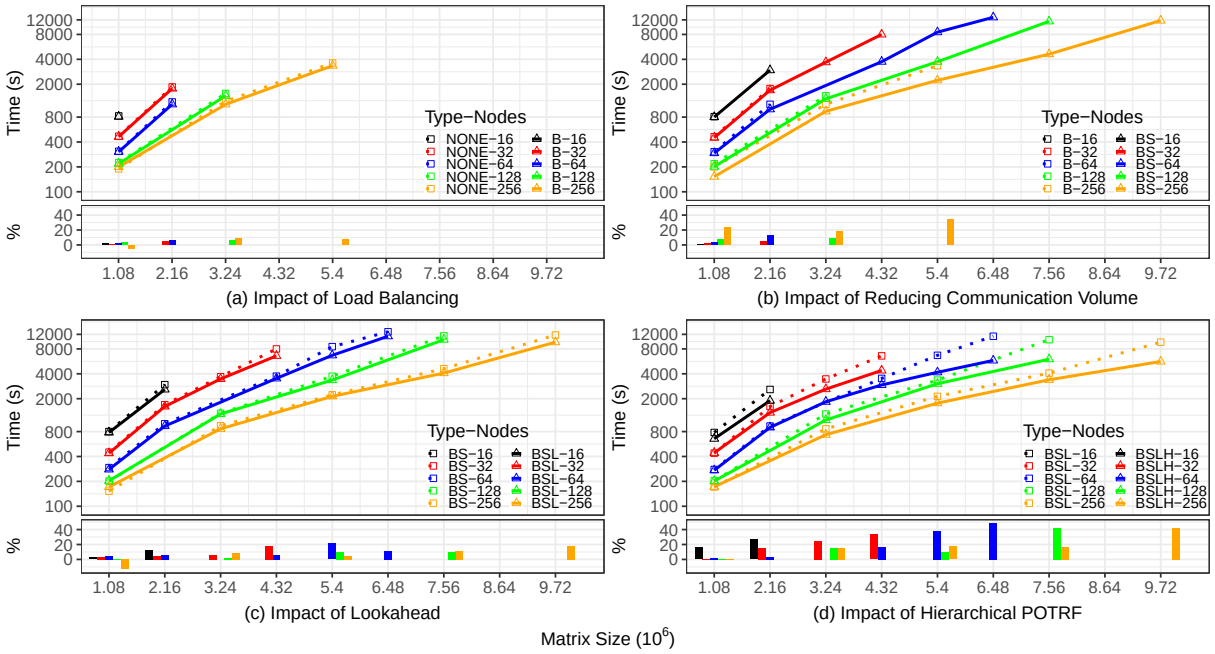


Figure 4: The incremental effect of the proposed optimizations for st-3D-sqexp. The bottom figures represent the respective resulting improvement as a percentage.

6.3.4 Hierarchical POTRF. Hierarchical *POTRF* creates a node-local taskpool that decomposes the diagonal tile into smaller sub-tiles and promote nested parallelism. This nested parallelism can then be executed on more local cores to speed up execution time of *POTRF* and reduce the critical path. Figure 4d depicts the efficiency of hierarchical *POTRF* **BSLH** compared to **BSL** for **st-3D-sqexp**. When scaling the matrix size up to the available memory limit, the performance improves almost by one-third. Because of the inherent characteristics of TLR Cholesky,the optimal tile size increases as the matrix size increases. This leads to the undesirable increase in the critical path, as larger tile size increases the execution time of kernels on dense tiles (*POTRF* and *SYRK*), which translates into longer critical path's time to solution. The use of hierarchical *POTRF* will increase the available parallelism and substantially cut down on the critical path execution time, leading to the significant improvement we are witnessing for larger matrix sizes.

6.3.5 Overall Effect of Proposed Optimizations. Figure 5 shows the effect of enabling all proposed optimizations compared to initial results without optimization for **st-3D-sqexp**, which gives us the whole picture. In this figure, both performance and memory footprint improve substantially, which create opportunities for large-scale experiments.

6.4 Extreme-Scale Runs

Figure 6 presents the extreme-scale results with matrix sizes up to 42M geospatial locations and using 16, 32, 64, 128, 256, 1024 and 4096 nodes for **st-3D-sqexp** with tile size 10000, 10000, 10000, 9000, 10000, 10800 and 10800 respectively, and 16, 32, 64, 128, 256, 512 and 1024 nodes for **st-2D-sqexp** with tile size 9000, 7200, 9000, 9000, 9000, 9000 and 10800 respectively. Each point in the plot corresponds to the factorization time for the largest matrix that can be factorized on a specific number of nodes according to the memory available on all the nodes involved. This setting may seem like a

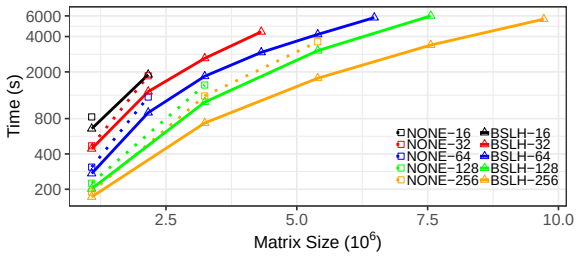


Figure 5: The effect of enabling all proposed optimization techniques for st-3D-sqexp.

weak scalability experiment, but in low-rank approximation weak scaling experiments result in a change of the ranks of tiles, and therefore, the number of operations and the memory necessary to store the low-rank matrix. In this figure, each point is associated with three properties: (1) number of nodes, (2) matrix size, and (3) execution time, which shows the performance and scalability for a certain number of nodes. For **st-2D-sqexp**, it can go up to a $32M$ matrix size on 512 nodes, almost 10× larger than previously reported with HiCMA, and up to $42M$ matrix size on $1,024$ nodes using $32,000$ cores. For **st-3D-sqexp**, which requires even more memory and computation (due to higher ranks and more rank disparity as we move further away from the diagonal), the results are presented up to $42M$ geospatial locations on $4,096$ nodes with a total of $130,000$ cores. These particular matrix sizes are appealing target for computational statisticians as this scale represents realistic workload datasets, but could not have been reached before. It should be noticed that in the presented setting the extreme-scale experiment has taken nearly 24 hours to complete. To put this elapsed time into perspective, let us compare against a dense Cholesky factorization on the same matrix size and number of nodes. The most time-consuming kernel is *GEMM*, running in a distributed setting at 80% of the theoretical peak performance. For the considered system, the sustained performance according to Top500 is 3.7 PFLOP/s on 4,096 nodes. Given that the number of FLOPs for a dense Cholesky factorization is $1/3\ N^3$, it would have taken approximately 77 days to compute the dense Cholesky factorization on a $42M$ matrix size, as opposed to slightly less than a single day for TLR Cholesky factorization, with the same, 10^{-9}, order threshold of the solution.

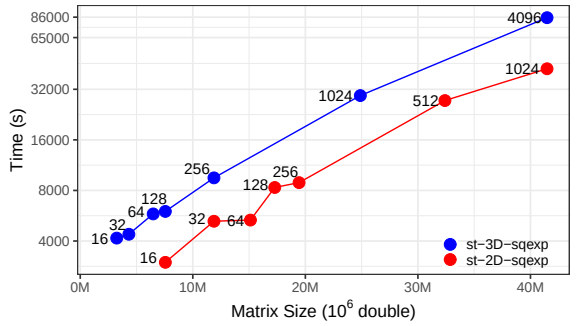


Figure 6: The performance results for the largest matrices that fit in memory for st-2D-sqexp and st-3D-sqexp.

7 Conclusion and Future Work

This paper presents the process to accommodate a heterogeneous workload from low-rank matrix computations over a task-based runtime, PaRSEC. In particular, we demonstrated that using the TLR compression data format together with four algorithmic improvements supported by a nimble task-based runtime, the TLR Cholesky factorization—which is the most time-consuming computational phase of the geospatial statistics approach for environmental applications—can be leveraged at an unprecedented scale. In addition to analyzing the setup in the context of a 2D statistical applications, we have highlighted the effectiveness and scalability of TLR Cholesky factorization on a 3D covariance matrix kernel at scales never reached before, $42M$ matrix size using $130,000$ cores. We believe the impact of these features goes well beyond the TLR compression data format for dense problems and may be directly applied to sparse direct solvers [10]. For future work, we plan to collaborate with domain scientists and compare our solution against first principles physics approaches using real 3D datasets. We would also investigate mixed precision techniques with TLR Cholesky (e.g., FP64, FP32, and FP16) by leveraging the data sparsity patterns for tiles located near, mid, and far from the diagonal. This would open new opportunities on GPUs for further performance gain.

Acknowledgments

This research was supported in part 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. The authors would like also to thank Cray Inc. and Intel in the context of the Cray Center of Excellence and Intel Parallel Computing Center awarded to the Extreme Computing Research Center at KAUST. For computer time, this research used the *Shaheen-2* supercomputer hosted at the Supercomputing Laboratory at KAUST.

References

[1] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes. 2018. ExaGeoStat: A High Performance Unified Software for Geostatistics on Manycore Systems. *IEEE Transactions on Parallel and Distributed Systems* 29, 12 (Dec 2018), 2771–2784.

[2] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes. 2018. Parallel Approximation of the Maximum Likelihood Estimation for the Prediction of Large-Scale Geostatistics Simulations. In *2018 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 98–108.

[3] E. Agullo, O. Aumage, M. Faverge, N. Furmento, F. Pruvost, M. Sergent, and S. Thibault. 2017. Achieving High Performance on Supercomputers with a Sequential Task-based Programming Model. *IEEE Transactions on Parallel and Distributed Systems* (2017).

[4] E. Agullo, J. Demmel, J. Dongarra, B. Hadri, J. Kurzak, J. Langou, H. Ltaief, P. Luszczek, and S. Tomov. 2009. Numerical Linear Algebra on Emerging Architectures: The PLASMA and MAGMA Projects. *Journal of Physics: Conference Series* 180 (2009).

[5] K. Akbudak, H. Ltaief, A. Mikhalev, A. Charara, A. Esposito, and D. E. Keyes. 2018. Exploiting Data Sparsity for Large-Scale Matrix Computations. In *Euro-Par 2018: Parallel Processing*, M. Aldinucci, L. Padovani, and M. Torquati (Eds.). Springer International Publishing, Cham, 721–734.

[6] K. Akbudak, H. Ltaief, A. Mikhalev, and D. Keyes. 2017. 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, 22–40.

[7] S. Ambikasaran and E. Darve. 2013. An $O(N \log N)$ Fast Direct Solver for Partial Hierarchically Semiseparable Matrices. *Journal of Scientific Computing* 57, 3 (2013), 477–501.

[8] P. Amestoy, C. Ashcraft, O. Boiteau, A. Buttari, J.-Y. L'Excellent, and C. Weisbecker. 2015. Improving Multifrontal Methods by Means of Block Low-Rank Representations. *SIAM Journal on Scientific Computing* 37, 3 (2015), A1451–A1474.

[9] P. R. Amestoy, A. Buttari, J.-Y. L'Excellent, and T. Mary. 2019. Performance and Scalability of the Block Low-Rank Multifrontal Factorization on Multicore Architectures. *ACM Trans. Math. Softw.* 45, 1, Article 2 (Feb. 2019), 26 pages.

[10] P. R. Amestoy, I. S. Duff, J.-Y. L'Excellent, and J. Koster. 2001. *MUMPS: A General Purpose Distributed Memory Sparse Solver.* Springer Berlin Heidelberg, Berlin, Heidelberg, 121–130. https://doi.org/10.1007/3-540-70734-4_16

[11] A. Aminfar, S. Ambikasaran, and E. Darve. 2016. A Fast Block Low-Rank Dense Solver with Applications to Finite-Element Matrices. *J. Comput. Phys.* 304 (2016), 170–188.

[12] E. Anderson, Z. Bai, C. H. Bischof, L. Susan Blackford, J. W. Demmel, J. J. Dongarra, J. J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. C Sorensen. 1999. *LAPACK User's Guide* (3rd ed.). SIAM, Philadelphia.

[13] C. Augonnet, S. Thibault, R. Namyst, and P. Wacrenier. 2011. StarPU: A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures. *Concurrency Computat. Pract. Exper.* 23 (2011), 187–198.

[14] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken. 2012. Legion: Expressing Locality and Independence with Logical Regions. In *International Conference for High Performance Computing, Networking, Storage and Analysis, SC*.

[15] M. Bebendorf. 2008. *Hierarchical Matrices: A Means to Efficiently Solve Elliptic Boundary Value Problems.* Lecture Notes in Computational Science and Engineering, Vol. 63. Springer. 269 pages.

[16] P. Beckman, K. Iskra, K. Yoshii, and S. Coghlan. 2006. Operating System Issues for Petascale Systems. *SIGOPS Operating Systems Review* 40, 2 (2006), 29–33.

[17] L.S. Blackford, J. Choi, A. Cleary, E.F. D'Azevedo, J.W. Demmel, I.S. Dhillon, J.J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D.W. Walker, and R.C. Whaley. 1997. *ScaLAPACK Users' Guide.* Society for Industrial and Applied Mathematics, Philadelphia. https://doi.org/10.1137/1.9780898719642

[18] R.D. Blumofe, C.F. Joerg, B.C. Kuszmaul, C.E. Leiserson, K.H. Randall, and Y. Zhou. 1996. Cilk: An Efficient Multithreaded Runtime System. *J. Parallel and Distrib. Comput.* 37, 1 (1996), 55 – 69. https://doi.org/10.1006/jpdc.1996.0107

[19] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, A. Haidar, T. Hérault, J. Kurzak, J. Langou, P. Lemarinier, H. Ltaief, P. Luszczek, A. YarKhan, and J. Dongarra. 2011. Flexible Development of Dense Linear Algebra Algorithms on Massively Parallel Architectures with DPLASMA. In *IPDPS Workshops*. IEEE, 1432–1441. http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6008655

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

[21] G. Bosilca, A. Bouteiller, A. Danalis, M. Faverge, T. Herault, and J.J. Dongarra. 2013. PaRSEC: Exploiting Heterogeneity to Enhance Scalability. *Computing in Science Engineering* 15, 6 (Nov 2013), 36–45. https://doi.org/10.1109/MCSE.2013.98

[22] A. Brandt. 1991. Multilevel Computations of Integral Transforms and Particle Interactions with Oscillatory Kernels. *Computer Physics Communications* 65, 1-3 (1991), 24–38.

[23] Q. Cao, Y. Pei, T. Herault, K. Akbudak, A. Mikhalev, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra. 2019. 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, 25–32.

[24] E. Chan, E.S. Quintana-Ortí, G. Quintana-Ortí, and R. van de Geijn. 2007. Supermatrix Out-of-order Scheduling of Matrix Operations for SMP And Multi-core Architectures. In *SPAA '07: Proceedings of the nineteenth annual ACM symposium on Parallel algorithms and architectures*. ACM, New York, NY, USA, 116–125. https://doi.org/10.1145/1248377.1248397

[25] A. Danalis, G. Bosilca, A. Bouteiller, T. Herault, and J. Dongarra. 2014. PTG: An Abstraction for Unhindered Parallelism. *Proceedings of WOLFHPC 2014: 4th International Workshop on DSLs and High-Level Frameworks for High Performance Computing*, 21–30. https://doi.org/10.1109/WOLFHPC.2014.8

[26] J. Dokulil, M. Sandrieser, and S. Benkner. 2016. Implementing the Open Community Runtime for Shared-Memory and Distributed-Memory Systems. *Proceedings - 24th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing, PDP 2016*, 364–368. https://doi.org/10.1109/PDP.2016.81

[27] A. Duran, R. Ferrer, E. Ayguadé, R.M. Badia, and J. Labarta. 2009. A Proposal to Extend the OpenMP Tasking Model with Dependent Tasks. *International Journal of Parallel Programming* 37, 3 (2009), 292–305.

[28] R. Garg and P. De. 2006. Impact of Noise on Scaling of Collectives: An Empirical Evaluation. In *HiPC'06: Proceedings of International Conference on High Performance Computing (LNCS)*, Springer (Ed.), Vol. 4297. 460–471.

[29] C. J. Geoga, M. Anitescu, and M. L. Stein. 2019. Scalable Gaussian Process Computations Using Hierarchical Matrices. *Journal of Computational and Graphical Statistics* 0, 0 (2019), 1–11. https://doi.org/10.1080/10618600.2019.1652616

[30] L. Greengard and V. Rokhlin. 1987. A Fast Algorithm for Particle Simulations. *J. Comput. Phys.* 73, 2 (1987), 325–348.

[31] N. Halko, P.-G. Martinsson, and J. A. Tropp. 2011. Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions. *SIAM Rev.* 53, 2 (2011), 217–288.

[32] R. Hoque, T. Herault, G. Bosilca, and J. Dongarra. 2017. 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 (ScalA '17)*. ACM, New York, NY, USA, Article 6, 8 pages. https://doi.org/10.1145/3148226.3148233

[33] H. Jagode, A. Danalis, and J. Dongarra. 2017. Accelerating NWChem Coupled Cluster through Dataflow-Based Execution. *The International Journal of High Performance Computing Applications* (01-2017 2017), 1–13.

[34] R. Kriemann. 2013. $\mathcal{H}$-LU Factorization on Many-core Systems. *Computing and Visualization in Science* 16, 3 (2013), 105–117.

[35] X. Lacoste, M. Faverge, G. Bosilca, P. Ramet, and S. Thibault. 2014. Taking Advantage of Hybrid Systems for Sparse Direct Solvers via Task-Based Runtimes. In *IEEE International Parallel & Distributed Processing Symposium Workshops (IPDPSW)*. 29–38. https://doi.org/10.1109/IPDPSW.2014.9

[36] H. Ltaief, A. Charara, D. Gratadour, N. Doucet, B. Hadri, E. Gendron, S. Feki, and D. Keyes. 2018. Real-Time Massively Distributed Multi-object Adaptive Optics Simulations for the European Extremely Large Telescope. In *IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 75–84.

[37] V. Martinez, F. Dupros, M. Castro, and P. Navaux. 2017. Performance Improvement of Stencil Computations for Multi-core Architectures based on Machine Learning. *Procedia Computer Science* 108, Supplement C (2017), 305 – 314. https://doi.org/10.1016/j.procs.2017.05.164 International Conference on Computational Science, ICCS 2017, 12-14 June 2017, Zurich, Switzerland.

[38] T. Mary. 2017. *Block Low-Rank Multifrontal Solvers: Complexity, Performance, and Scalability.* Ph.D. Dissertation. Paul Sabatier University, Toulouse, France.

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

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

[41] R.G. Parr. 1980. Density Functional Theory of Atoms and Molecules. In *Horizons of Quantum Chemistry*, Kenichi Fukui and Bernard Pullman (Eds.). Springer Netherlands, Dordrecht, 5–15.

[42] G. Peano. 1890. Sur une courbe, qui remplit toute une aire plane. *Math. Ann.* 36, 1 (1890), 157–160.

[43] Y. Pei, G. Bosilca, I. Yamazaki, A. Ida, and J. Dongarra. 2019. Evaluation of Programming Models to Address Load Imbalance on Distributed Multi-Core CPUs: A Case Study with Block Low-Rank Factorization. In *PAW-ATM Workshop at SC19*. ACM, ACM, Denver, CO.

[44] J. Reinders. 2010. *Intel Threading Building Blocks Outfitting C++ for Multi-core Processor Parallelism.* O'Reilly Media.

[45] F.-H. Rouet, X.S. Li, P. Ghysels, and A. Napov. 2016. A Distributed-Memory Package for Dense Hierarchically Semi-Separable Matrix Computations Using Randomization. *ACM Trans. Math. Software* 42, 4, Article 27 (June 2016), 35 pages.

[46] M. L. Stein. 2014. Limitations on Low Rank Approximations for Covariance Matrices of Spatial Data. *Spatial Statistics* 8 (2014), 1 – 19. https://doi.org/10.1016/j.spasta.2013.06.003 Spatial Statistics Miami.

[47] Y. Sun and M.L. Stein. 2016. Statistically and Computationally Efficient Estimating Equations for Large Spatial Datasets. *Journal of Computational and Graphical Statistics* 25, 1 (2016), 187–208.

[48] M. Tillenius, E. Larsson, E. Lehto, and N. Flyer. 2013. A Task Parallel Implementation of a Scattered Node Stencil-based Solver for the Shallow Water Equations. In *Proc. 6th Swedish Workshop on Multi-Core Computing :*. Halmstad University, 33–36.

[49] S.J. Treichler. 2014. *Realm: Performance Portability through Composable Asynchrony.* Ph.D. Dissertation. Stanford University.

[50] D. Tsafrir, Y. Etsion, D.G. Feitelson, and S. Kirkpatrick. 2005. System Noise, OS Clock Ticks, and Fine-grained Parallel Applications. In *ICS '05: Proceedings of the 19th Annual International Conference on Supercomputing*. ACM Press, New York, NY, USA, 303–312.

[51] E. E. Tyrtyshnikov. 1996. Mosaic-Skeleton Approximations. *Calcolo* 33, 1 (1996), 47–57. https://doi.org/10.1007/BF02575706

[52] A. G. Wilson and H. Nickisch. 2015. Kernel Interpolation for Scalable Structured Gaussian Processes (KISS-GP). In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37 (ICML'15)*. JMLR.org, 1775–1784. http://dl.acm.org/citation.cfm?id=3045118.3045307

[53] W. Wu, A. Bouteiller, G. Bosilca, M. Faverge, and J. Dongarra. 2015. Hierarchical DAG Scheduling for Hybrid Distributed Systems. In *2015 IEEE International Parallel and Distributed Processing Symposium*. 156–165.

[54] J. Xia, Y. Xi, and M. Gu. 2012. A Superfast Structured Solver for Toeplitz Linear Systems via Randomized Sampling. *SIAM J. Matrix Anal. Appl.* 33, 3 (2012), 837–858. https://doi.org/10.1137/110831982 arXiv:https://doi.org/10.1137/110831982

[55] C.D. Yu, S. Reiz, and G. Biros. 2018. Distributed-memory Hierarchical Compression of Dense SPD Matrices. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis (SC '18)*. IEEE Press, Piscataway, NJ, USA, Article 15, 15 pages.