2023 IEEE International Conference on Cluster Computing (CLUSTER) | 979-8-3503-0792-4/23/$31.00 ©2023 IEEE | DOI: 10.1109/CLUSTER52292.2023.00035

Reducing Data Motion and Energy Consumption of Geospatial Modeling Applications Using Automated Precision Conversion

Qinglei Cao[1,2,4,5], Sameh Abdulah[3,6], Hatem Ltaief[3,6], Marc G. Genton[3,6], David Keyes[3,6], and George Bosilca[2,5]

[1]Saint Louis University, US
[2]University of Tennessee, US
[3]King Abdullah University of Science and Technology, KSA
[4]{*qinglei.cao*}@*slu.edu*
[5]{*qcao, bosilca*}@*icl.utk.edu*
[6]{*sameh.abdulah, hatem.ltaief, marc.genton, david.keyes*}@*kaust.edu.sa*

***Abstract*—The burgeoning interest in large-scale geospatial modeling, particularly within the domains of climate and weather prediction, underscores the concomitant critical importance of accuracy, scalability, and computational speed. Harnessing these complex simulations' potential, however, necessitates innovative computational strategies, especially considering the increasing volume of data involved. Recent advancements in Graphics Processing Units (GPUs) have opened up new avenues for accelerating these modeling processes. In particular, their efficient utilization necessitates new strategies, such as mixed-precision arithmetic, that can balance the trade-off between computational speed and model accuracy. This paper leverages PaRSEC runtime system and delves into the opportunities provided by mixed-precision arithmetic to expedite large-scale geospatial modeling in heterogeneous environments. By using an automated conversion strategy, our mixed-precision approach significantly improves computational performance (up to 3X) on Summit supercomputer and reduces the associated energy consumption on various Nvidia GPU generations. Importantly, this implementation ensures the requisite accuracy in environmental applications, a critical factor in their operational viability. The findings of this study bear significant implications for future research and development in high-performance computing, underscoring the transformative potential of mixed-precision arithmetic on GPUs in addressing the computational demands of large-scale geospatial modeling and making a stride toward a more sustainable, efficient, and accurate future in large-scale environmental applications.**

***Index Terms*—Task-based runtime, Geospatial statistics, Automated precision conversion, GPU acceleration, HPC**

I. Introduction

Modern computing architectures have evolved to become increasingly powerful, driven by the need to meet the ever-growing computational demands of various applications. These architectures often combine diverse hardware components, such as multi-core CPUs and accelerators (e.g., GPUs), along with intricate memory hierarchies and interconnects. This complexity presents challenges in terms of resource utilization, load balancing, data management, etc. In this context, task-based runtime systems [1]–[6] are designed to decompose applications into smaller, independent tasks that can be executed concurrently, allowing for greater adaptability and flexibility in managing the diverse hardware components found in modern architectures. By dynamically scheduling tasks across different processing elements, task-based runtime systems can optimize resource utilization and exploit the inherent parallelism available in these complex computing environments, which can lead to improved performance, efficiency, and scalability for a wide range of applications [7]–[11]. As computing systems continue to evolve towards more complex and heterogeneous designs, the importance of task-based runtime systems in managing and optimizing these environments will only grow.

Geospatial modeling is an interdisciplinary field that involves the creation and analysis of spatial data representations to simulate, predict, and understand complex phenomena occurring in the physical world. It plays a crucial role in addressing global challenges and enhancing our understanding of Earth's systems, which are essential for studying the impacts of climate change on ecosystems, species distribution, natural resources, etc. Despite numerous efforts [12]–[15], geospatial statistics still faces several challenges that must be addressed to fully harness its potential. One of the primary challenges is efficiently handling the large-scale, high-resolution spatial data that these models often require. Therefore, developing scalable algorithms and leveraging advanced computing architectures are essential for addressing these data-intensive challenges and ensuring that geospatial models can effectively manage the increasing volume and complexity of spatial information.

At the frontier of these two topics there is a growing interest in integrating geospatial modeling with task-based runtime systems [16]–[18] aimed at tackling the multifaceted challenges intrinsic to this field. These studies exploit the benefits of task-based runtime systems, covering a broad spectrum of topics such as scalable algorithms, efficient data management, and high user productivity. Notably, previous research [18] put forth a framework that unites geospatial modeling applications with the PaRSEC runtime system [6] using adaptive mixed-precision computations on CPU architectures.

2168-9253/23/$31.00 ©2023 IEEE
DOI 10.1109/CLUSTER52292.2023.00035

This paper builds on this existing framework, by extending it to support heterogeneous architectures and also by introducing an innovative automated precision conversion strategy within PaRSEC. The basic principles of task scheduling and execution were not affected, allowing PaRSEC to maintain its strong asynchronous capabilities. However, details of how those principles are applied differ significantly and typically transitioning from CPU to GPU necessitates careful thought, new solutions, and sometimes even a different architectural approach. For instance:

- efficiently managing data transfer between host and devices, or between devices themselves, is a critical challenge, and poor handling of these transfers can lead to GPU idly waiting for data, and thus to lower efficiency.
- GPU operates asynchronously with respect to CPU, and therefore ensuring correct program execution often requires careful coordination and synchronization between tasks on CPU and GPU. This can be particularly challenging in a dynamic task-based runtime where tasks may have complex dependencies.
- GPUs have their own separate memory, which requires specialized memory management, not only for handling temporary memory usage, but also for ensuring data consistency between different tasks.

From an application perspective, we accelerate the dense mixed-precision Cholesky factorization that drives Gaussian maximum log-likelihood estimation (MLE) in geospatial modeling. We assess the impact of the automated conversion strategy on data movement and MLE's elapsed time on Summit supercomputer. We also report significant reduction in energy consumption on various Nvidia GPU generations (i.e., V100, A100, and H100). More importantly, our method maintains the required accuracy of the application driver by verifying the accuracy of the statistical parameter estimators that govern the modeling of environmental applications. This is paramount to the success of the approach, as inaccurate results, regardless of speed or efficiency, would ultimately prove useless in real-world applications.

Our main contributions are as follows:

- adding support for adaptive mixed-precision computations on heterogeneous architectures in PaRSEC;
- reducing data transfers by relying on an automated precision conversion strategy;
- validating the parameter estimations accuracy via synthetic geospatial datasets under various configurations;
- evaluating the proposed approach on Summit supercomputer with up to 384 V100 GPUs;
- investigating the impact on power consumption using various Nvidia GPU generations, i.e., V100, A100, and H100.

The remainder of the paper is organized as follows. Section II covers related work, and Section III provides the essential background knowledge. We motivate the impact of mixed precisions in Section IV by benchmarking the most time-consuming kernel (i.e., GEMM) of the Cholesky factorization. Section V describes the adaptive mixed-precision Cholesky framework in the context of heterogeneous architectures. Section VI discusses the automated precision conversion strategy. Section VII evaluates the accuracy, performance, and power consumption and we conclude in Section VIII.

II. Related Work

A. Runtime Systems

Task-based runtime systems have gained significant attention in recent years due to their ability to efficiently manage the complexity and parallelism of modern architectures. These systems focus on expressing parallelism through fine-grained tasks, enabling efficient scheduling and load balancing while hiding communication and synchronization overheads. QUARK (QUeueing And Runtime for Kernels) [19], [20] targets shared-memory multicore architectures and provides a simple and lightweight API for expressing fine-grained tasks and their dependencies. Its primary focus is on linear algebra operations and has been successfully used to implement several high-performance numerical libraries, such as PLASMA [21]. OpenMP [3] is an industry-standard API for shared-memory parallel programming. With the introduction of tasking features in recent versions, OpenMP has evolved into a task-based runtime system that supports dynamic scheduling and data dependencies and allows users to express parallelism through directives and runtime library routines, enabling the compiler and runtime system to generate and manage tasks efficiently. OmpSs [2] extends the OpenMP standard with support for heterogeneous architectures, asynchronous parallelism, and data dependencies. Moreover, COMP Superscalar (COMPSs) [22] is designed to simplify the development of applications for distributed infrastructures and offers a programming interface for application development and a runtime system that leverages the intrinsic parallelism of applications during execution. StarPU [1] provides a unified framework for distributed heterogeneous multicore architectures, allowing developers to write parallel code by annotating computational kernels with their respective task implementations while the runtime takes care of scheduling and data movement. HPX (High-Performance ParalleX) [4] is a general-purpose C++ runtime system designed for parallel and distributed computing. It is based on the ParalleX execution model. Legion [5] is a data-centric task-based runtime system designed for distributed and heterogeneous architectures. It introduces a novel programming model that decouples the specification of tasks and data from the mapping of tasks onto hardware resources. This separation allows for the automatic discovery of parallelism and efficient management of data locality, leading to improved performance and scalability.

B. Mixed-Precision Matrix Computations

Mixed-precision computations have gained significant attention [23]–[26] and has been widely applied in various domains, including deep learning [27], [28], computational fluid dynamics [29], astronomy [30], and quantum mechanics [31]. The driving innovation behind all these techniques is the same, leverage mixed-precision hardware features to

accelerate computations while maintaining the required level of accuracy.

Mixed-precision computations have been employed in climate and weather applications to improve modeling and prediction over large spatial regions. In [12], [13], a band-based approach was utilized to accelerate matrix operations, exploiting the band data sparsity pattern of the matrix operator. By representing correlations between distant locations in lower precision, the computational efficiency of modeling and prediction operations was enhanced. This approach builds upon the covariance tapering methods used in statistics [14], [15]. Recently, Higham and Mary conducted a linear algebra survey in [32] to explore existing mixed-precision algorithms. They proposed a formula based on the norm at the block-level computation to select the appropriate precision for mixed-precision computations. This approach helps guide the choice of precision to balance accuracy and computational efficiency. Cao et al. [18] utilized the aforementioned formula to scale MLE operations on the Fugaku supercomputer, toying with up to three different arithmetic precisions. In this study, we aim to take this study further, and accelerate geospatial statistical applications by leveraging adaptive mixed-precision technology on heterogeneous architectures and deploying an automated precision conversion strategy within a task-based runtime, PaRSEC.

Lower precision computations also offer the advantage of reduced energy consumption, making it an attractive approach for fostering energy efficiency in computing operations. In [33], energy-efficient matrix operations were proposed in the context of mixed-precision linear solvers with iterative refinement. The study reported significant energy savings. This paper also delves into energy efficiency linear solvers across various Nvidia GPU architectures, but without having to store copies of the whole matrix in considered precisions.

III. Background

A. Geospatial Modeling

Gaussian processes (GPs) are widely used in machine learning and Bayesian inference as flexible and powerful tool for modeling complex systems and making predictions. GPs can be applied in many applications, including spatial statistics, where GPs are used in modeling spatial data because of their easy-to-implement structure to model spatially correlated data, where the values of nearby locations are expected to be more similar than those of distant locations. Under the assumption of stationarity, i.e., constant statistical properties over the entire spatial domain, GPs can be used to model a wide range of spatial phenomena, such as temperature, rainfall, and air pollution, and to make predictions at unobserved locations.

Maximum likelihood estimation (MLE) is a widespread technique utilized for modeling geospatial data when combined with GPs. The GP model defines mean and covariance functions between a given set of spatial locations that help evaluate the likelihood function given a set of parameters. The MLE operation involves finding the values of the model parameters that maximize the likelihood function, which measures the probability of observing the given spatial data under the assumed model. Assume a collection of observations $\mathbf{Z}$ derived from a Gaussian random field $Z(\mathbf{s})$, where $\mathbf{Z} = \{Z(\mathbf{s}_1), \ldots, Z(\mathbf{s}_n)\}^\top$ represents the values at n spatial locations $\mathbf{s}_1, \ldots, \mathbf{s}_n$ in $\mathbb{R}^d$. For GP modeling, we assume a stationary and isotropic Gaussian random field characterized by a mean of zero. The covariance function $C(\mathbf{h}; \boldsymbol{\theta}) = \text{cov}\{Z(\mathbf{s}), Z(\mathbf{s}+\mathbf{h})\}$ is utilized, where $\mathbf{h} \in \mathbb{R}^d$ denotes a lag vector, and $\boldsymbol{\theta} \in \mathbb{R}^q$ represents an unknown parameter vector. Herein, the covariance function $C(\mathbf{h}; \boldsymbol{\theta})$ relies on the distance between any two locations, denoted by $\boldsymbol{\Sigma}(\boldsymbol{\theta})$. This discrete covariance matrix is defined with entries $\boldsymbol{\Sigma}_{ij} = C(\mathbf{s}_i - \mathbf{s}_j; \boldsymbol{\theta})$ for $i, j = 1, \ldots, n$. $\boldsymbol{\Sigma}(\boldsymbol{\theta})$ is a symmetric and positive definite matrix. Statistical inference about $\boldsymbol{\theta}$ is often based on the Gaussian log-likelihood function:

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

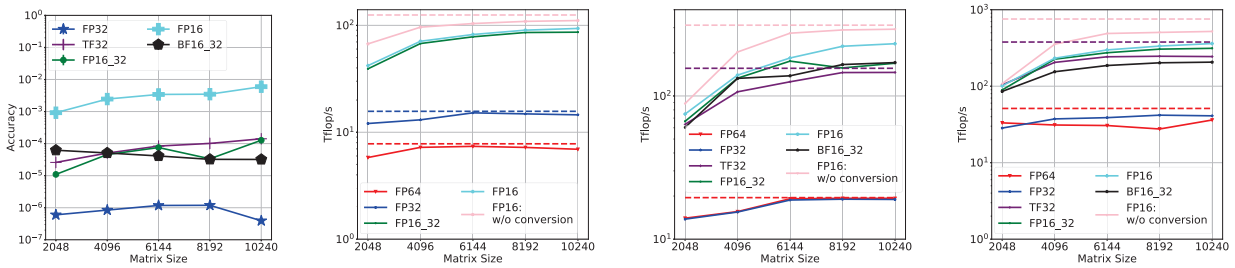
The modeling process depends on defining the covariance function $C(\mathbf{h}; \boldsymbol{\theta})$ to compute $\widehat{\boldsymbol{\theta}}$, the parameter vector that maximizes (1). In this study, we consider two existing functions, which we summarize as:

*1) The 2D/3D Squared Exponential Covariance Function (***2D/3D-sqexp):** $C(\mathbf{h}; \boldsymbol{\theta}) = \sigma^2 \exp(-h^2/\beta)$ where $h = \|\mathbf{h}\|$ represents the distance between spatial locations, σ^2 is the variance parameter, β is the range parameter representing a strong, medium, or weak correlation between the spatial locations, and $\boldsymbol{\theta} = (\sigma^2, \beta)^\top$.

*2) The 2D Matérn Covariance Function (***2D-Matérn):** The Matérn covariance function is a flexible family of covariance functions that incorporates different levels of smoothness. It is defined as $C(\mathbf{h}; \boldsymbol{\theta}) = \sigma^2(2^{1-\nu}/\Gamma(\nu))(h/\beta)^\nu \mathcal{K}_\nu(h/\beta)$ where σ^2 is the variance parameter, β is the range parameter, ν controls the smoothness representing a rough or smooth field, and $\mathcal{K}_\nu(\cdot)$ is the modified Bessel function of the second kind of order ν, and $\boldsymbol{\theta} = (\sigma^2, \beta, \nu)^\top$.

B. The PaRSEC Runtime System

PaRSEC [6] is a powerful and versatile task-based platform designed to facilitate the efficient execution of fine-grained tasks on distributed many-core heterogeneous architectures. Its primary objective is to provide high-performance solutions for a wide range of applications in scientific computing and other domains requiring high computational throughput. Like most task-based runtime systems, by representing algorithms as Directed Acyclic Graphs (DAGs), with vertices symbolizing tasks and edges defining dependencies between them, PaRSEC enables effective management of task scheduling and data movement. One of the most prominent features of PaRSEC is its asynchronous, architecture-aware scheduling, which allows tasks to be executed as soon as their dependencies are satisfied, in contrast to synchronous scheduling, which requires tasks to be executed in a predefined order. Asynchronous scheduling enables PaRSEC to make better use of available computational resources by avoiding unnecessary idle time and reducing the



(a) Accuracy. (b) Performance, V100 (NVLink). (c) Performance, A100 (SXM). (d) Performance, H100 (PCIe).

Fig. 1: Accuracy and performance of GEMM benchmark on 1 GPU. The dashed line is the theoretical peak of the corresponding solid line about that precision format. FP64 operates on tensor cores on A100 and H100, so they share the same peak performance. Regarding accuracy, lower is better, while for performance, higher is better.

overall execution time. Furthermore, the architecture-aware nature of the scheduling process ensures that tasks are optimally scheduled across heterogeneous resources, taking into account their unique characteristics, such as processing capabilities.

PaRSEC employs a rich set of Domain-Specific Languages (DSLs) to facilitate interaction with the runtime system. These DSLs provide increased flexibility, enabling domain scientists to express algorithms more productively and allowing for a more straightforward mapping of the algorithm to the underlying hardware. One such DSL, the Parameterized Task Graph (PTG) [34], used in this paper, represents task dependencies through a concise yet comprehensive task graph description called Job Data Flow (JDF). The resulting DAG can be viewed as a collection of task classes containing information necessary for creating and executing task instances, including operations to be performed on different computational units. The Template Task Graph (TTG) [35] offers a C++ API and expands on the PTG concept by generalizing parameter types and enabling data-dependent selection of task dependencies. Other DSLs, like Dynamic Task Discovery [36], are less domain science-oriented and express DAG through sequential task insertion in nested loops, which might encounter similar scalability issues as seen with other distributed task-insertion runtimes like StarPU and QUARK.

IV. Mixed-Precision HPC Applications: A Must!

Nvidia's V100, A100, and H100 GPUs have been specifically engineered to support a range of precision levels in order to meet diverse computational requirements and cater to the unique demands of deep learning applications. *But what if we could steer these specific hardware features for HPC applications?* Table I outlines the theoretical peak performance across different floating-point precision formats of V100, A100, and H100. This table highlights the escalating performance gains that can be achieved when transitioning from the traditional double or single precision modes to Tensor Core-accelerated modes, thereby paving the way for substantial enhancements in mixed-precision computation performance. Within the context of the tile Cholesky factorization required by MLE, the majority of floating-point operations (typically more than 90%) are attributable to the General Matrix Multiply kernel (GEMM, $C = alpha \times A \times B + beta \times C$). Consequently, we assess the accuracy and performance achieved on three types of Nvidia GPUs listed in Table I, focusing specifically on GEMM benchmark. This evaluation allows us to better understand the technology trend, the potential benefits and limitations of mixed-precision computations over the course of several GPU generations.

TABLE I: Peak performance of Nvidia GPUs (Tflop/s).

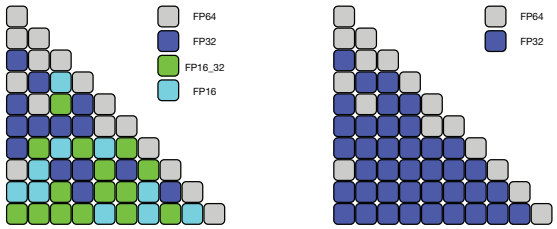
Precision	V100 (NVLink)	A100 (SXM)	H100 (PCIe)
FP64	7.8	9.7	25.6
FP64 Tensor	-	19.5	51.2
FP32	15.7	19.5	51.2
TF32 Tensor	-	156	378
FP16 Tensor	125	312	756
BF16 Tensor	-	312	756

In this study, we extend the standard GEMM benchmark (which includes cudaMemcpyAsync from host to device, GEMM execution on the device, and cudaMemcpyAsync from device to host) to encompass multiple supported precisions on Nvidia GPUs. These include FP64, FP32, TF32, FP16_32 (where A and B are in FP16 while C and operation are in FP32), BF16_32 (with A and B in BF16 while C and operation in FP32), and FP16 (A, B, C, and operation are all in FP16). We insert timers as needed and compare the accuracy of GEMMs with different precisions to FP64 GEMM using the Frobenius norm. Fig. 1 presents the GEMM accuracy and performance of various precisions on V100, A100, and H100 GPUs with the data randomly initialized. With regard to performance results, data movement between host and device is not considered, while datatype conversion (existing in FP16_32, BF16_32, and FP16) is accounted unless otherwise specified. The findings from these figures indicate that (1) each precision format displays distinct characteristics in terms of accuracy and performance - notably, TF32, FP16_32, and BF16_32 exhibit similar behaviors, a finding corroborated in [24] and (2) near-theoretical peak performance is achieved for each precision when ignoring the cost of data movement and datatype conversion. Additionally, the volume of data transferred between the host and device varies based on different precisions. Therefore, considering the combined

effects of accuracy and performance on these Nvidia GPUs, we incorporate FP64, FP32, FP16_32, and FP16 into our adaptive-precision framework. This approach can capture benefits offered by different precisions on modern GPUs, resulting in a more versatile and efficient computational framework.

V. Adaptive Multi-precisions Cholesky

Within the context of the covariance matrix, which is formulated through MLE, a noteworthy pattern emerges where the correlation among geospatial locations that are significantly distant from one another diminishes as one moves farther away from the matrix diagonal. This can be captured during the computation process through the utilization of multiple precision levels. In order to illustrate this, we refer to Fig. 2a. This figure demonstrates the precision map at the tile level that is adopted for the numerical kernels, which are responsible for carrying out the operations on each tile. For instance, when computations are performed on tiles that are situated along the diagonal, they invariably make use of FP64 precision since these tiles contain the strongest correlations. Operation-precisions applied on off-diagonal tiles are adaptively determined by a tile-centric strategy, which is determined by thresholding the ratio of the Frobenius norms (F-norm) of the tile and the global matrix [32] based on the application-dependent accuracy, in accordance with the methodology on CPUs outlined in [18], i.e., $||A_{ij}|| \times NT/||A|| \leq u_{req}/u_{low}$. $||A_{ij}||$ is the F-norm at tile level; NT, the number of tiles in a dimension; $||A||$, F-norm of the whole matrix; u_{req}, the application required accuracy; u_{low}, machine epsilon of a given lower precision. In our study, we employ a Monte Carlo arithmetic method to check the impact of reduced precision on the application. This involves utilizing different accuracy levels to decrease the precision of the matrix tiles, which subsequently affects the accuracy of parameter estimation. Therefore, this u_{req} is application-dependent. Simply put, if the ratio of the F-norm of a tile to the global matrix is relatively smaller, a lower level of precision can be adopted for operations on that particular tile. In this paper, our exploration of u_{low} extends now to a variety of floating-point precision formats that are compatible with GPUs, including FP64, FP32, FP16_32, and FP16, as illustrated in Section IV. Powered by tensor cores, the resulting faster computations may further displace the bottleneck to data movement overheads. We have made the choice to ignore BF16_32 because the performances



(a) Kernel execution. (b) Data storage.

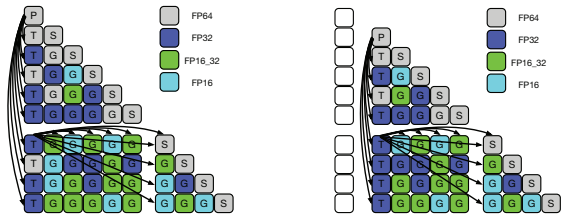
Fig. 2: Precision map of kernel execution and data storage.

Algorithm 1: Adaptive GPU-based MP Cholesky.

```
for k = 0 to NT − 1
    DPOTRF (k, k, C_kk)
    for m = k + 1 to NT − 1
        TRSM (m, k, C*_kk, C_mk)
        ← DPOTRF (k, k, C*_kk)
    for m = k + 1 to NT − 1
        DSYRK (m, k, C*_mk, C_mm)
        ← TRSM (m, k, C_kk, C*_mk)
    for m = k + 2 to NT − 1
        for n = k + 1 to m − 1
            GEMM (m, n, k, C*_mk, C*_nk, C_mn)
            ← TRSM (m, k, C_kk, C*_mk)
            ← TRSM (n, k, C_kk, C*_nk)
```

in BF16_32 and FP16_32 are similar in the considered GPU cards, although BF16_32 has a smaller machine epsilon [25], [37].

Algorithm 1 describes the sequential process of the adaptive mixed-precision (MP) Cholesky on block- or tile- partitioned matrices. This algorithm involves the utilization of four distinct numerical kernels, namely POTRF (Cholesky factorization), TRSM (triangular solve), SYRK (symmetric rank-k update), and GEMM (general matrix multiply). The arrow $\leftarrow$ associated with the blue task demonstrates the dependencies that exist between tasks which engender data movement. The asterisk symbol $*$ on the parameter signifies that data is involved in that communication. The character "D" prepended to POTRF and SYRK numerical kernels means FP64 arithmetics since they operate on diagonal tiles. The other two kernels can run on any precisions depending on the tile position. To provide a more comprehensive understanding, Fig. 3 offers a visual demonstration of the execution of tasks during the first two iterations as described in Algorithm 1 along with representative communications introduced by POTRF $\rightarrow$ TRSMs, TRSM $\rightarrow$ GEMMs and TRSM $\rightarrow$ SYRK. Each communication process possesses its own unique datatype, the specifics of which will be expounded upon in the subsequent section. It is important to highlight that the formats FP16_32 and FP16 are exclusively supported in GEMM kernel when using Nvidia GPUs. As a consequence of this hardware limitation, TRSM must execute in FP32 in case the precision format of that particular tile turns



(a) Panel $k = 0$. (b) Panel $k = 1$.

Fig. 3: Demonstration of the first two iterations in Algorithm 1. P, POTRF; T, TRSM; S, SYRK; G, GEMM. Arrows are representative dependencies introducing communications. The white color indicates numerical kernels on that tile are finished.

out to be FP16_32 or FP16. This results in those tiles being stored directly in FP32 during the matrix generation phase, and Fig. 2b visually represents this concept by showing the corresponding precision in data storage as it applies to the example provided in Fig. 2a.

VI. Automated Precision Conversion Strategy

Section V outlines communication patterns in Cholesky factorization: (1) POTRF (k, k) on diagonal tile (k, k) broadcasts to TRSMs in column k, and (2) TRSM (m, k) broadcasts to GEMMs in row m, GEMMs in column m, and SYRK (m, k). Given the adaptive tile-centric precision strategy, the precision of the data received by a task may not match the precision it operates on, necessitating a likely precision conversion. This datatype conversion can occur either at the sender's end, referred to as sender/source task conversion (STC), or at the receiver's end, called receiver/target task conversion (TTC). The latter was originally introduced in [18], [38]. As their names suggest, STC ensures that the data sent matches the receiver's precision requirements up front; while TTC works in a way where the sender task forwards the data in the precision it generates, and the receiver is in charge of the local conversion of the data according to its specific needs. Two potential performance advantages of the innovative STC strategy over TTC [18], [38] are evident. Using TRSM $\rightarrow$ GEMMs as an example, the datatype conversion will only occur once in TRSM rather than in each successive GEMM, provided what TRSM sends matches what GEMMs require. We recall the overhead of datatype conversion in Fig. 1. In addition to this, if down-casting occurs in TRSM, the succeeding GEMMs can transfer data of lower precision to GPUs, mitigating data motion overheads. Table II presents the time required for different precision on one V100 GPU (within Summit supercomputer) when moving a tile/matrix to GPU and then applying a GEMM operation. All numerical kernels in our framework operate on GPUs, so these scenarios can exist in our approach. As can be seen from the table, data movement can obliterate any GEMM performance on the GPU, particularly if a GEMM task operates in a lower precision than what the data has been moved into the GPU (such as FP16 while receiving data in FP64). This overhead could be further amplified when considering the two input data in GEMM, as shown in Algorithm 1.

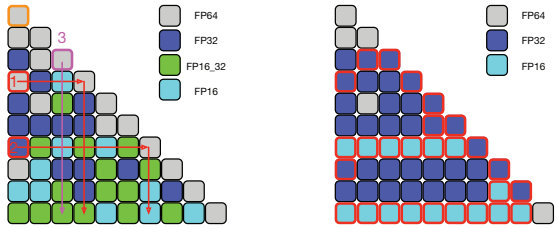
We emphasize that while consistently downgrading to the lowest precision could further reduce GPU data transfer, it might also unnecessarily compromise the accuracy, given that subsequent tasks may function at higher precisions. Furthermore, we refrain from constantly sending relevant data to all subsequent tasks since it may necessitate maintaining and propagating several precisions of each data. For instance, consider the tile (0, 0) outlined in orange in Fig. 4a: POTRF (0, 0) applied on this tile need to retain (for TRSMs on the same process) and broadcast (for TRSMs on a different process) the same data in three precisions (FP64, FP32, and FP16).

We provide an approach that combines STC and TTC, automatically adapting our strategy during runtime when initiating communication to reduce data transfer to GPUs. This prevents unnecessary accuracy loss and avoids the redundancy of multiple precision data. In essence, we opt for STC when all successors of a sender operate at a lower precision; otherwise, TTC is deployed. The precision in STC is determined by the highest precision among all successors, while the precision in TTC is based on the precision of kernel execution in POTRF or TRSM that issues communications.

Fig. 4a illustrates this automated precision conversion strategy by offering three instances of STC applied in the same example discussed in Fig. 2a. Tiles marked with the numbers 1 and 2 denote TRSMs operating on these tiles in the first iteration ($k = 0$) in Algorithm 1, and POTRF (2, 2) operates on the tile labeled with number 3 in the third iteration ($k = 2$). Tasks (POTRF and TRSMs) operating on these tiles can send data in lower precision than the kernel-generated precision since all their successors work with lower-precision numerical kernels. The recipient task/tasks might still require conversion, but as previously mentioned in this section, we strive to minimize the transfer cost by always lowering, as much as allowed/possible, the precision of the data about to be transferred. Fig. 4b enumerates precisions of all tiles, thereby determining the communication precision for tasks executed on these tiles. Algorithm 2 details the sequential process for creating the communication-precision map (Fig. 4b) based on the kernel-precision map (Fig. 4a), which traverses successors of POTRF and TRSM tasks. The main objective is to identify tasks that are suitable for STC. During this traversal, every tile's $comm\_precision$ is initialized to the lowest precision (STC) and concludes when TTC advantages are observed. NT

TABLE II: Time measurement on V100 (milliseconds).

Matrix Size	2048	4096	6144	8192	10240
Move one tile/matrix in FP64	**0.67**	**2.68**	**6.04**	**10.74**	**16.78**
Move one tile/matrix in FP32	0.34	1.34	3.02	5.37	8.39
Move one tile/matrix in FP16	0.17	0.67	1.51	2.68	4.19
Execute GEMM in FP64	2.2	17.62	59.47	140.96	275.32
Execute GEMM in FP32	1.09	8.75	29.54	70.03	136.78
Execute GEMM in FP16	**0.14**	**1.1**	**3.71**	**8.8**	**17.18**



(a) Demonstration of STC. (b) Precision in communication.

Fig. 4: Automated precision conversion. (a) showcases when STC is applied. (b) depicts the precision in the communication at tile level when POTRF or TRSM operates on this tile; tasks on tiles with red boundaries use STC, and others TTC.

Algorithm 2: Automated precision conversion.

```
/* Precision of kernel operated on each tile */
Input: kernel precision
/* Precision of communication issued on each tile */
Output: comm_precision
/* Diagonal tiles (k, k) operating POTRF (k, k) */
for k = 0 to NT − 1
    comm_precision(k, k) = FP_32
    for m = k + 1 to NT − 1
        if kernel_precision(m, k) == FP_64
            comm_precision(k, k) = FP_64
            break
/* Off-diagonal tiles (m, k) operating TRSM (m, k) */
for k = 0 to NT − 2
    for m = k to NT − 1
        comm_precision(m, k) = FP_16
        storage_precision(m, k) =
          get_storage_precision(kernel_precision(m, k))
        /* Check row broadcast in TRSM */
        for n = k + 1 to m
            comm_precision(m, k) =
              get_higher_precision(comm_precision(m, k),
              kernel_precision(m, n))
            if comm_precision(m, k) is not lower than
              storage_precision(m, k)
                comm_precision(m, k) = storage_precision(m, k)
                goto done_this_tile
        /* Check column broadcast in TRSM */
        for n = m + 1 to NT − 1
            comm_precision(m, k) =
              get_higher_precision(comm_precision(m, k),
              kernel_precision(n, m))
            if comm_precision(m, k) is not lower than
              storage_precision(m, k)
                comm_precision(m, k) = storage_precision(m, k)
                goto done_this_tile
        done_this_tile:
```

represents the number of tiles in a dimension, and the time complexity of this entire operation is $O((NT)^3)$. A notable aspect of this procedure is that each task's calculation (POTRF and TRSM) is independent, indicating the potential for parallelization. Upon successful execution of the algorithm, the outcome is provided to the runtime via the DSL. In this way, the precision of each communication can be automatically ascertained at runtime, thereby, on the receiving end, reducing the amount of data transfer to GPUs, avoiding unnecessary datatype conversions as much as possible, while preventing unnecessary accuracy loss, and eliminating redundancy of multiple precision data.

VII. Performance Results and Analysis

A. Experimental Settings

The experiments are conducted on three systems with different Nvidia GPUs, as described in Table I.

- Summit at Oak Ridge National Laboratory (ORNL). A GPU-based IBM system including 4,356 compute nodes. Each node contains two 22-core Power9 CPUs operating at 3.07 GHz and 256 GB of main memory, and each CPU is equipped with three NVIDIA Tesla V100 GPUs [39]. The deployed CUDA version is 11.0.3.
- Guyot at Innovative Computing Laboratory (ICL). A GPU-based AMD compute node. It contains two EPYC 7742 64-Core CPUs at 2.25GHz, 2063 GB of main memory, and eight NVIDIA A100-SXM4-80GB GPUs. The deployed CUDA version is 11.4.4.
- Haxane at ICL. A GPU-based Intel compute node. It contains two 8-core Xeon(R) Silver 4309Y CPUs at 2.80 GHz, 63 GB of main memory, and one NVIDIA H100 PCIe GPU. The deployed CUDA version is 12.1.1.

For the data distribution, we deploy a process grid $P \times Q$ (as square as possible) where $P \leq Q$. Precision formats of storage, calculation, and communication are adaptively determined using the tile-centric strategy, detailed in Sections V and VI. The authors in [23] show FP16_32 has a lower error bound than FP16, so we experimentally determine its machine epsilon in applications. We recall that FP64 on A100 and H100 achieve the same theoretical peak performance as FP32 due to tensor cores. The optimized tile size is determined empirically and set to 2048. We run our experiments multiple times, and once no noticeable performance variability is measured, the maximal performance is reported. The execution time of Algorithm 2 is less than 0.1 seconds in all experiments, which is relatively negligible compared to the cost of Cholesky factorization.

B. Accuracy Evaluation in Geospatial Modeling Applications

The goal of GP modeling is to estimate the unknown parameters, denoted as $\boldsymbol{\theta}$, of a statistical model with a given covariance function. Once these parameters are estimated, the model can be utilized for predicting future measurements with unknown values.

We employ Monte Carlo simulations to assess the parameter estimation capabilities of the MLE operation. We created 100 synthetic datasets for varying parameter configurations and subsequently applied the MLE algorithm to the generated datasets. Our objective was to recover the original parameters $\boldsymbol{\theta}$ that were used to generate the datasets. These synthetic datasets closely resemble real-world data encountered in climate and weather applications. In the literature, Monte Carlo simulation has been employed to evaluate various implementations of the MLE algorithm, particularly in cases where an approximation is made for the covariance matrix [40]–[43].

Our aim is to assess the effectiveness of MP linear algebra solvers when employed for the MLE function described in (1). Referring back to Section III-A, we focus on two covariance functions: squared exponential and Matérn. In our experiments, we use 100 replicas of 40,000 synthetic locations and their corresponding measurement vectors for each configuration. For optimization, we employ the BOBYQA algorithm from the NLOPT library [44], setting the optimization tolerance to 10^{-9}. All parameters are constrained within the range of 0.01 to 2, with the optimization algorithm consistently initiating from the lower bound values.

The experiments focus on exploring extreme settings regarding the correlation β between locations in both the squared exponential and Matérn covariance functions, as well as the data smoothness ν in the Matérn covariance functions. Using the provided synthetic datasets, the correlation β can range

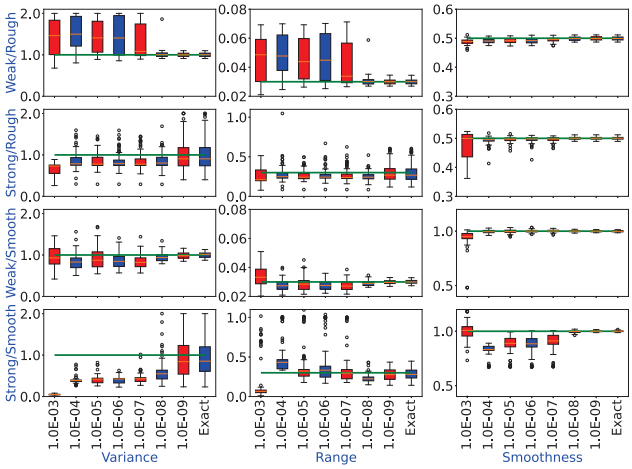


Fig. 5: Parameter estimations for 2D synthetic datasets with two levels of correlation, i.e., weak and strong, and two levels of smoothness, i.e., rough and smooth, using different levels of mixed-precision accuracy. The solid green line is the actual value of that parameter.

from weak ($\beta = 0.03$) to strong ($\beta = 0.3$), while the smoothness can vary from rough ($\nu = 0.5$) to smooth ($\nu = 1$). It is expected that real-world datasets may also exhibit similar extremes, enabling us to effectively evaluate the estimation accuracy of the MP-based MLE algorithm.

In Fig. 5, the boxplots display the estimated parameters of the 2D datasets using two covariance functions: squared exponential (rows 1 and 2; **2D-sqexp**) and Matérn (rows 1, 2, 3, and 4; **2D-Matérn**). It is shown for both covariance functions that an accuracy of 10^{-9} provides the most accurate results compared to the exact computation. Achieving an accuracy of 10^{-9} yields highly precise estimations, rendering the need for oversolving via exact computation unnecessary. Regarding the squared exponential kernels, an accuracy of 10^{-4} can still be employed, achieving a satisfactory level of application accuracy. Nevertheless, this study considers the resulting estimation difference to be acceptable, as it is not

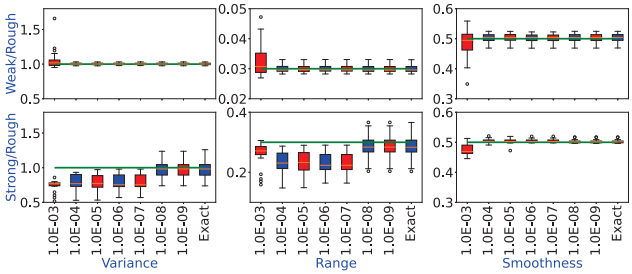
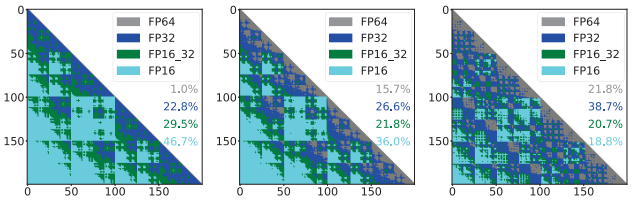


Fig. 6: Parameter estimations for 3D synthetic datasets with two levels of correlation, i.e., weak and strong, using different levels of mixed-precision accuracy. The solid green line is the actual value of that parameter.



(a) **2D-sqexp**. (b) **2D-Matérn**. (c) **3D-sqexp**.

Fig. 7: Kernel precision executed on each tile with matrix size 409,600 and tile size 2048. The percentage of the number of tiles in each precision is marked in the respective color.

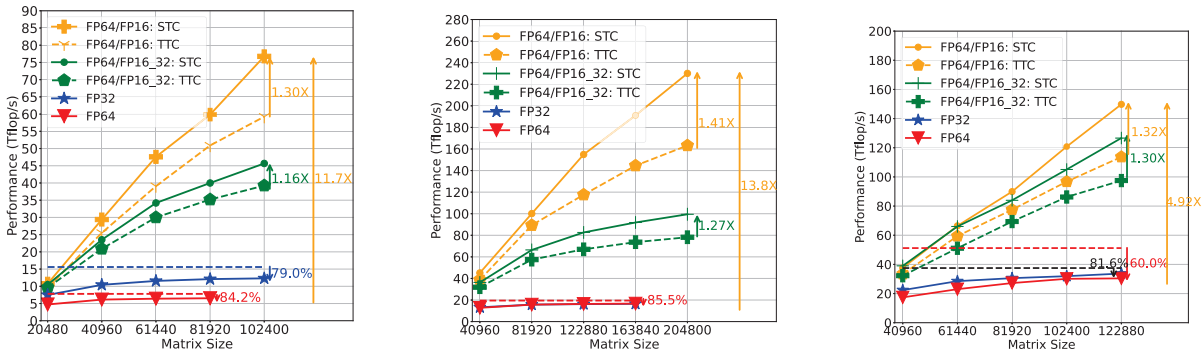
significantly large. In the case of the Matérn covariance function, only the accuracy of 10^{-9} can meet the required level of precision. Fig. 6 presents the boxplots illustrating the estimated parameters for 3D datasets when using the squared exponential covariance function (**3D-sqexp**). The results demonstrate that an accuracy of 10^{-8} yields estimations that are highly close to the exact solution.

C. Precision Map of Kernel Execution

Fig. 7 provides a visualization of the precision of numerical kernels executed on each individual tile, as referred to Fig. 2a, by presenting a heatmap in the context of the three applications that are previously discussed. The accuracy portrayed in Fig. 7 is maintained at the required accuracy threshold for each application, i.e., 10^{-4} for **2D-sqexp**, 10^{-9} for **2D-Matérn** and 10^{-8} for **3D-sqexp**. From these figures, each application exhibits its unique characteristics: **2D-sqexp** is the most cost-effective, with 29.5% and 46.7% of tiles executing kernels in FP16_32 and FP16 respectively, while **3D-sqexp** is the most resource-intensive, with over 60% of tiles executing kernels in either FP64 or FP32. These create varying levels of load imbalance in terms of computation, communication, and memory footprint, highlighting the versatility and adaptability of our approach.

D. Impact of Automated Precision Conversion Strategy

As mentioned in Section VI, STC sends fewer data than TTC, thus reducing data movement between CPU and GPU on the receiver end, but STC cannot always be applied. Therefore, automated precision conversion entails a dynamic strategy of whether STC can be utilized during the runtime when communication is initiated. In a bid to effectively illustrate the benefits of this approach, Fig. 8 presents a comparative analysis of the performance of STC and TTC under two extreme conditions, FP64/FP16_32, and FP64/FP16, on one V100 GPU of Summit, one A100 GPU of Guyot, and one H100 GPU of Haxane. This can also provide insights into the performance that can be achieved with each precision. Regarding the kernel execution map (refer to Fig. 2a) specific to these two settings, numerical kernels on diagonal tiles and off-diagonal tiles operate in FP64 and in FP16_32 (for the FP64/FP16_32 configuration) or FP16 (for the FP64/FP16 configuration), respectively. In this case, all communications can employ the



(a) One V100 on Summit (b) One A100 on Guyot (c) One H100 on Haxane

Fig. 8: Performance of precision conversion strategies and efficiency on one GPU. The dashed red/blue lines are the theoretical peak of that precision. The black dashed line in (c) is the sustained GEMM performance (FP64) achieved by the GEMM benchmark in Fig. 1d. FP64 operates on tensor cores on A100 and H100, thus achieving the same theoretical peak than FP32.

STC strategy. Communications can be classified as either STC or TTC. Therefore, if all communications fall under TTC, it can be viewed as a lower bound in performance, while all communications in STC as an upper limit for performance improvement. For a real-world application, performance likely lies between these two extreme conditions.

This figure presents numerous observations and findings.

- High efficiency with FP64 and FP32 computations. On the V100 platform, our implementation is able to achieve 84.2% and 79% of the theoretical peak performance using FP64 and FP32, respectively. Similarly, on A100, the performance exceeded 85% for both FP64 and FP32 operations. On H100 with PCIe, we attain around 62% of the theoretical peak performance. This slightly diminished performance on the H100 can be attributed to the fact that the achieved practical GEMM performance is marginally lower compared to that on V100 and A100, as illustrated in Fig. 1d. Nevertheless, when the performance on H100 is juxtaposed with the practical GEMM performance, it surpasses 82%.
- Significant improvements when transitioning from TTC to STC. As anticipated, decreasing the amount of data to be transferred (by decreasing the precision of the transferred data) has an extremely positive impact. Indeed, the performance of STC surpasses that of TTC in all instances. This led to a speedup of up to 1.3X on V100, 1.41X on A100, and 1.27X on H100.

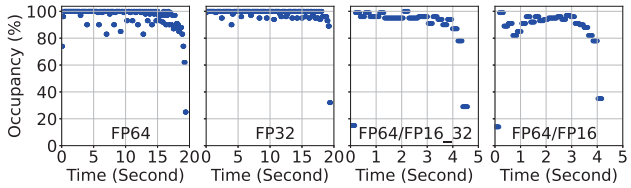


Fig. 9: GPU Occupancy of one H100 on Haxane.

- Substantial gains when shifting from FP64 to lower precision computations. In particular, the F64/FP16 configuration can achieve the best performance in our framework, which leads to a speedup of more than 11X to FP64 on both the V100 and A100 platforms. Especially on A100, this speedup is quite notable when compared to the theoretical performance speedup of the FP16 tensor core to the FP64 tensor core (16X, referring to Table I). On H100, this speedup was slightly lower, coming in at 4.7X. This lesser speedup can be attributed to the limited main memory size on Haxane, which is 63 GB (see Section VII-A), thereby imposing a restriction on the matrix size that can be experimented with.

In order to gain a better understanding of the execution on H100, Fig. 9 provides a depiction of the GPU occupancy for STC on the largest matrix size as shown in Fig. 8c. Each data point plotted within these figures represents the actual time occupancy. This was measured at regular interval, employing the tools provided by Nvidia. A careful examination of these figures reveals that 100% GPU utilization can be routinely attained when dealing with FP64 and FP32, which means no data transfers are holding up computations, so all data transfers are completely overlapped with computations. When considering FP64/FP16_32 and FP64/16, it is observed that the GPU occupancy regularly exceeds 80%. This indicates that even in these scenarios, a significant portion of GPU's computational resources is being effectively used, further demonstrating the efficiency of the execution on H100.

Similarly to Fig. 8, Fig. 11 presents the performance results gathered from a single node that utilizes multiple GPUs on Summit and Guyot. Upon comparing Fig. 11 with Fig. 8, our implementation is highly proficient at virtually reaching linear scalability when transitioning from the use of one GPU to the employment of multiple GPUs within a single physical node. In addition, we manage to (1) maintain a significant efficiency rate of over 80% for both FP64 and FP32 computations, as compared to their respective theoretical peaks; (2) facilitate a

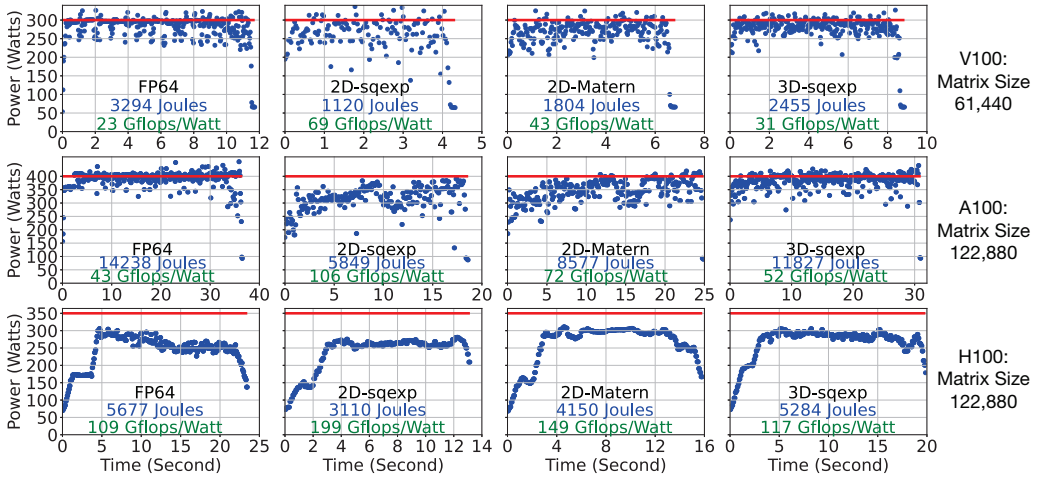
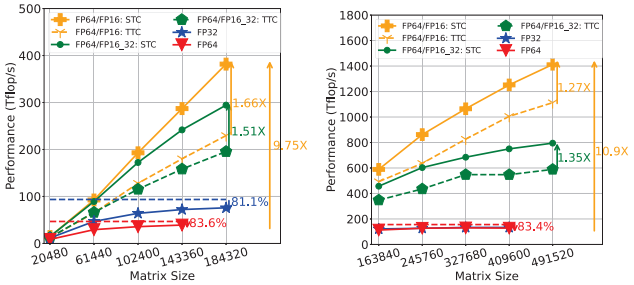


Fig. 10: Power consumption of Cholesky in FP64, the proposed MP approach for **2D-sqexp**, **2D-Matérn**, and **3D-sqexp**. The first/second/third rows are on one V100/A100/H100 GPU, respectively. The solid red line is the max TDP on that GPU.



(a) Six V100 GPUs. (b) Eight A100 GPUs.

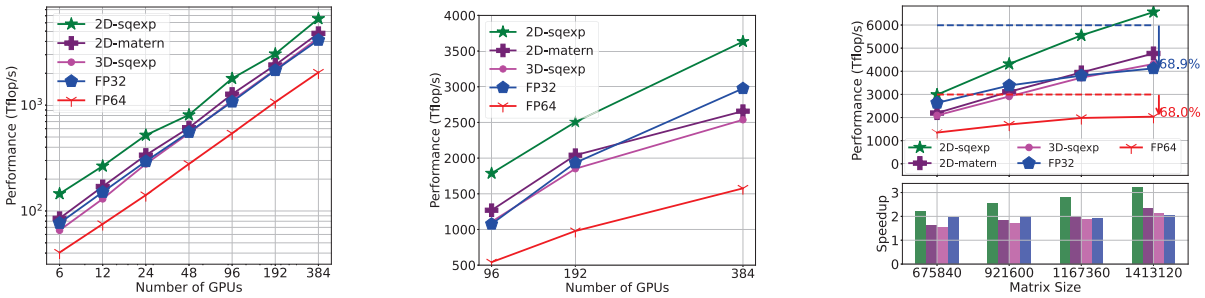
Fig. 11: Performance of precision conversion strategies and efficiency on one Summit/Guyot-node. The dashed red/blue lines are the theoretical peak of that precision.

considerable speedup ratio of up to 1.66X when transitioning from TTC to STC operations; (3) exhibit a remarkable speedup of 9.75X on the Summit and an even higher speedup of 10.9X on Guyot when switching from FP64 to FP64/FP16 calculations.

E. Power Consumption

Power consumption has increasingly become a critical factor in high-performance computing, particularly as systems grow in scale. The pursuit of computational speed and performance must now be balanced with the need for energy efficiency, as the environmental and economic costs of power-hungry operations have become untenable. Fig. 10 presents a comparative analysis of energy consumption on one V100 Summit, one A100 Guyot, and one H100 Haxane. This comparison is made between our proposed mixed-precision approach for **2D-sqexp**, **2D-Matérn**, and **3D-sqexp**, and their respective FP64 counterpart. The data points, representing energy measurements taken at regular intervals using provided Nvidia tools, sometimes exceed the max Thermal Design Power (TDP). This could potentially be attributable to measurement errors, ambient noise, or short-term power spikes. The matrix size utilized on V100 is the largest one that fits in GPU memory using F64 (i.e., $61,440$), while the matrix size of $122,880$ on both A100 and H100 is determined by the main memory size on Haxane. Therefore, given the cubic time complexity of Cholesky factorization, theoretically, the energy consumption on V100 would be tripled if the matrix size on V100 is the same as on A100/H100.

Fig. 10 also encapsulates total energy consumption in joules and performance achieved per watt in Gflops/Watt on single NVIDIA GPU hardware generation. It is evident from the figure that the proposed mixed-precision approach with the automated conversion STC, which allows calculations to be executed in lower precision where high precision is not necessary, can considerably reduce the required computational resources and, consequently, the energy consumed. This leads not only to decreased power consumption but also to an augmented computational capacity of the system, as more operations can be performed within the same power budget. To be more specific, on A100 and H100 architectures, tensor cores are employed for FP64 computations, while FP32 operations utilize regular cores because of accuracy. As a result, the performance and energy consumption trends for A100 and H100 for FP64 and FP32 are similar. This leads to the proposed mixed-precision approach yielding lesser energy savings on A100/H100 compared to V100, especially for **3D-sqexp** where a large proportion of tiles operate in FP32 (see Fig. 7). Furthermore, the majority of higher precision tiles tend to cluster near the diagonal, resulting in an even smaller proportion of kernels in lower precision, e.g., FP16. In addition, these figures also exemplify the progression in technology scaling from V100 to A100 to H100 in terms of performance and energy efficiency. It is noteworthy that



(a) Weak scalability. (b) Strong scalability. (c) MP effect on 64 nodes (384 GPUs).

Fig. 12: Performance evaluation on Summit. The dashed lines in (c) show the theoretical peak performance of that precision.

the real-time power consumption on H100 does not reach the max TDP, despite the occupancy consistently hitting 100%, as shown in Fig. 9. Assessment of energy efficiency of our mixed-precision approach on multiple GPU-based nodes is an interesting avenue for future work but will require advanced monitoring tools and a dedicated access on resources that are usually shared across users, e.g., the network interconnect.

F. Performance Scalability

We thoroughly test our mixed-precision approach associated with our automated precision conversion on Summit on three key aspects: weak scalability, strong scalability, and the effect of mixed-precision on FP64, as shown in Fig. 12. With respect to weak scalability, the problem size was systematically expanded with the number of GPUs. Results show near-linear scalability, suggesting that our methodology could efficiently adapt to incremental increases in computational resources. Next, we examine the strong scalability, where we maintain a constant matrix size $(798,720)$ while increasing the number of GPUs. The results demonstrate notable strong scalability, suggesting that our approach can effectively distribute a fixed-size problem across additional resources to expedite the computation process. Performance on 384 GPUs falls slightly short of linear scale, possibly due to running out of work, exacerbating higher communication and runtime overheads [45]. Lastly, we analyze the mixed-precision effect on 64 Summit nodes (i.e., 384 GPUs) by showing the actual performance and its speedup compared to FP64. The baseline performance of FP64 achieves 68.0% of the theoretical peak, which is a commendable result compared to the 74.0% LINPACK efficiency on Summit listed on Top500 [46]. As for the effect of mixed precisions with automated conversion, **2D-sqexp**, **2D-Matérn**, and **3D-sqexp** perform better than FP32 when matrix size increases, with up to 3.2X speedup compared to FP64.

An interesting observation across the three applications reveals that **3D-sqexp** achieves the least performance, while **2D-sqexp** demonstrates the highest performance thanks to its low-precision resilience. This aligns with our earlier discussions in Section VII-C. It is crucial to remember that the required accuracy significantly impacts the precision of the kernel operations carried out on each tile (refer to Section V), thereby influencing the resulting performance. As previously mentioned, we employ an accuracy level that is universally applicable across all scenarios in each application. In practice, technologies like sampling [47]–[49] can preprocess the dataset to catch such characteristics beforehand. This allows for the possibility of a higher precision threshold for these applications, which in turn, results in selecting more tiles near the critical path (diagonal and sub-diagonal tiles [16]) in lower precision and hence, achieving superior performance.

All in all, these findings demonstrate the versatility, efficiency, and scalability of the proposed approach on heterogeneous architectures in carrying out mixed-precision matrix operations using automated conversion, while still maintaining satisfactory accuracy for geostatistical modeling.

VIII. Conclusion and Future Work

This paper underscores the significance of employing mixed-precision arithmetic as a means to expedite large-scale geospatial modeling. In this context, we harness the capabilities of PaRSEC, a nimble task-based runtime system, which aids in establishing an adaptive mixed-precision framework with an automated precision conversion strategy on heterogeneous systems. Our proposed implementation not only improves performance but also reduces the associated storage cost and energy consumption, while still guaranteeing the required accuracy in environmental applications.

Looking ahead, we intend to further our research by developing an efficient GPU library and combining the strengths of mixed precisions with tile low-rank (TLR) computations [16], [17] to address the curse of dimensionality. We ultimately aim to improve the efficiency and performance of large-scale geospatial modeling tasks, with a particular focus on applications related to climate and weather prediction.

IX. 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. Regarding the computer time, this research utilized the Summit supercomputer at ORNL in US and the computing nodes at ICL in the University of Tennessee, Knoxville, US.

REFERENCES

[1] C. Augonnet, S. Thibault, R. Namyst, and P. Wacrenier, "StarPU: A unified platform for task scheduling on heterogeneous multicore architectures," *Concurrency Computat. Pract. Exper.*, vol. 23, pp. 187–198, 2011.

[2] A. Duran, R. Ferrer, E. Ayguadé, R. Badia, and J. Labarta, "A Proposal to Extend the OpenMP Tasking Model with Dependent Tasks," *International Journal of Parallel Programming*, vol. 37, no. 3, pp. 292–305, 2009.

[3] OpenMP, "OpenMP 5.2 Complete Specifications," 2021. [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf

[4] T. Heller, H. Kaiser, and K. Iglberger, "Application of the ParalleX Execution Model to Stencil-based Problems," *Computer Science - Research and Development*, vol. 28, no. 2-3, pp. 253–261, 2013.

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

[6] 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.

[7] A. Danalis, H. Jagode, G. Bosilca, and J. Dongarra, "PaRSEC in practice: Optimizing a legacy chemistry application through distributed task-based execution," in *2015 IEEE International Conference on Cluster Computing*, Sept 2015, pp. 304–313.

[8] H. Jagode, A. Danalis, and J. Dongarra, "Accelerating nwchem coupled cluster through dataflow-based execution," *The International Journal of High Performance Computing Applications*, p. 1–13, 01-2017 2017. [Online]. Available: http://journals.sagepub.com/doi/10.1177/1094342016672543

[9] Q. Cao, Y. Pei, T. Herauldt, 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)*. IEEE, 2019, pp. 25–32.

[10] T. Herault, Y. Robert, G. Bosilca, R. J. Harrison, C. A. Lewis, E. F. Valeev, and J. J. Dongarra, "Distributed-memory multi-gpu block-sparse tensor contraction for electronic structure," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 537–546.

[11] Q. Cao, R. Alomairy, Y. Pei, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, "A framework to exploit data sparsity in tile low-rank cholesky factorization," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 414–424.

[12] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Geostatistical modeling and prediction using mixed precision tile cholesky factorization," in *2019 IEEE 26th international conference on high performance computing, data, and analytics (HiPC)*. IEEE, 2019, pp. 152–162.

[13] S. Abdulah, Q. Cao, Y. Pei, G. Bosilca, J. Dongarra, M. G. Genton, D. E. Keyes, H. Ltaief, and Y. Sun, "Accelerating geostatistical modeling and prediction with mixed-precision computations: A high-productivity approach with parsec," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 4, pp. 964–976, 2021.

[14] R. Furrer, M. G. Genton, and D. Nychka, "Covariance tapering for interpolation of large spatial datasets," *Journal of Computational and Graphical Statistics*, vol. 15, no. 3, pp. 502–523, 2006.

[15] C. G. Kaufman, M. J. Schervish, and D. W. Nychka, "Covariance tapering for likelihood-based estimation in large spatial data sets," *Journal of the American Statistical Association*, vol. 103, no. 484, pp. 1545–1555, 2008.

[16] 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.

[17] Q. Cao, Y. Pei, K. Akbudak, G. Bosilca, H. Ltaief, D. Keyes, and J. Dongarra, "Leveraging parsec runtime support to tackle challenging 3d data-sparse matrix problems," in *2021 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2021, pp. 79–89.

[18] Q. Cao, S. Abdulah, R. Alomairy, Y. Pei, P. Nag, G. Bosilca, J. Dongarra, M. G. Genton, D. E. Keyes, H. Ltaief *et al.*, "Reshaping geostatistical modeling and prediction for extreme-scale environmental applications," in *2022 SC22: International Conference for High Performance Computing, Networking, Storage and Analysis (SC)*. IEEE Computer Society, 2022, pp. 13–24.

[19] A. YarKhan, "Dynamic Task Execution on Shared and Distributed Memory Architectures," Ph.D. dissertation, 2012. [Online]. Available: http://trace.tennessee.edu/utk_graddiss/1575

[20] A. YarKhan, J. Kurzak, and J. Dongarra, "QUARK Users' Guide: QUeueing And Runtime for Kernels," *University of Tennessee Innovative Computing Laboratory Technical Report ICL-UT-11-02*, 2011.

[21] A. YarKhan, J. Kurzak, P. Luszczek, and J. Dongarra, "Porting the plasma numerical library to the openmp standard," *International Journal of Parallel Programming*, vol. 45, pp. 612–633, 2017.

[22] F. Lordan, E. Tejedor, J. Ejarque, R. Rafanell, J. Alvarez, F. Marozzo, D. Lezzi, R. Sirvent, D. Talia, and R. M. Badia, "Servicess: An interoperable programming framework for the cloud," *Journal of grid computing*, vol. 12, no. 1, pp. 67–91, 2014.

[23] P. Blanchard, N. J. Higham, F. Lopez, T. Mary, and S. Pranesh, "Mixed precision block fused multiply-add: Error analysis and application to gpu tensor cores," *SIAM Journal on Scientific Computing*, vol. 42, no. 3, pp. C124–C141, 2020.

[24] W. Sun, A. Li, T. Geng, S. Stuijk, and H. Corporaal, "Dissecting tensor cores via microbenchmarks: Latency, throughput and numeric behaviors," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 1, pp. 246–261, 2022.

[25] A. Haidar, H. Bayraktar, S. Tomov, J. Dongarra, and N. J. Higham, "Mixed-precision iterative refinement using tensor cores on gpus to accelerate solution of linear systems," *Proceedings of the Royal Society A*, vol. 476, no. 2243, p. 20200110, 2020.

[26] A. Abdelfattah, S. Tomov, and J. Dongarra, "Investigating the benefit of fp16-enabled mixed-precision solvers for symmetric positive definite matrices using gpus," in *Computational Science–ICCS 2020: 20th International Conference, Amsterdam, The Netherlands, June 3–5, 2020, Proceedings, Part II*. Springer, 2020, pp. 237–250.

[27] C. Gong, Z. Jiang, D. Wang, Y. Lin, Q. Liu, and D. Z. Pan, "Mixed precision neural architecture search for energy efficient deep learning," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–7.

[28] S. Nandakumar, M. Le Gallo, C. Piveteau, V. Joshi, G. Mariani, I. Boybat, G. Karunaratne, R. Khaddam-Aljameh, U. Egger, A. Petropoulos *et al.*, "Mixed-precision deep learning based on computational memory," *Frontiers in neuroscience*, vol. 14, p. 406, 2020.

[29] A. Walden, E. Nielsen, B. Diskin, and M. Zubair, "A mixed precision multicolor point-implicit solver for unstructured grids on gpus," in *2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 2019, pp. 23–30.

[30] N. Doucet, H. Ltaief, D. Gratadour, and D. Keyes, "Mixed-precision tomographic reconstructor computations on hardware accelerators," in *2019 IEEE/ACM 9th Workshop on Irregular Applications: Architectures and Algorithms (IA3)*. IEEE, 2019, pp. 31–38.

[31] T. Boku, K.-I. Ishikawa, Y. Kuramashi, and L. Meadows, "Mixed precision solver scalable to 16000 mpi processes for lattice quantum chromodynamics simulations on the oakforest-pacs system," in *2017 Fifth International Symposium on Computing and Networking (CANDAR)*. IEEE, 2017, pp. 362–368.

[32] N. J. Higham and T. Mary, "Mixed precision algorithms in numerical linear algebra," *Acta Numerica*, vol. 31, pp. 347–414, 2022.

[33] A. Haidar, A. Abdelfattah, M. Zounon, P. Wu, S. Pranesh, S. Tomov, and J. Dongarra, "The design of fast and energy-efficient linear solvers: On the potential of half-precision arithmetic and iterative refinement techniques," in *Computational Science–ICCS 2018: 18th International Conference, Wuxi, China, June 11–13, 2018, Proceedings, Part I*. Springer, 2018, pp. 586–600.

[34] A. Danalis, G. Bosilca, A. Bouteiller, T. Herault, and J. Dongarra, "PTG: An Abstraction for Unhindered Parallelism," 2014, pp. 21–30.

[35] G. Bosilca, R. J. Harrison, T. Herault, M. M. Javanmard, P. Nookala, and E. F. Valeev, "The Template Task Graph (TTG)-an Emerging Practical Dataflow Programming Paradigm for Scientific Simulation at Extreme Scale," in *IEEE/ACM 5th International Workshop on Extreme Scale Programming Models and Middleware (ESPM2)*. IEEE, 2020.

[36] 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, 2017, pp. 6:1–6:8.

[37] M. Fasi, N. J. Higham, M. Mikaitis, and S. Pranesh, "Numerical behavior of nvidia tensor cores," *PeerJ Computer Science*, vol. 7, p. e330, 2021.

[38] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Geostatistical Modeling and Prediction Using Mixed Precision Tile Cholesky Factorization," in *2019 IEEE 26th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, 2019, pp. 152–162.

[39] Summit, "Summit User Guide," 2023. [Online]. Available: https://docs.olcf.ornl.gov/systems/summit_user_guide.html

[40] 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*. IEEE, 2018, pp. 98–108.

[41] ——, "ExaGeoStat: A High Performance Unified Software for Geostatistics on Manycore Systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 12, pp. 2771–2784, 2018.

[42] H. Huang and Y. Sun, "Hierarchical Low Rank Approximation of Likelihoods for Large Spatial Datasets," *J. of Computational and Graphical Statistics*, vol. 27, no. 1, pp. 110–118, 2018.

[43] H. Huang, S. Abdulah, Y. Sun, H. Ltaief, D. E. Keyes, and M. G. Genton, "Competition on spatial statistics for large datasets," *Journal of Agricultural, Biological and Environmental Statistics*, vol. 26, pp. 580–595, 2021.

[44] S. G. Johnson and J. Schueller, "Nlopt: Nonlinear optimization library," *Astrophysics Source Code Library*, pp. ascl–2111, 2021.

[45] E. Slaughter, W. Wu, Y. Fu, L. Brandenburg, N. Garcia, W. Kautz, E. Marx, K. S. Morris, Q. Cao, G. Bosilca *et al.*, "Task Bench: A parameterized Benchmark for Evaluating Parallel Runtime Performance," in *SC20: International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 2020.

[46] Top500, "Summit in Top500," 2023. [Online]. Available: https://www.top500.org/system/179397/

[47] F. Liang, Y. Cheng, Q. Song, J. Park, and P. Yang, "A resampling-based stochastic approximation method for analysis of large geostatistical data," *Journal of the American Statistical Association*, vol. 108, no. 501, pp. 325–339, 2013.

[48] M. H. Barbian and R. M. Assunçao, "Spatial subsemble estimator for large geostatistical data," *Spatial Statistics*, vol. 22, pp. 68–88, 2017.

[49] Y. Song, W. Dai, and M. G. Genton, "Large-scale low-rank gaussian process prediction with support points," *arXiv preprint arXiv:2207.12804*, 2022.