GPU-Accelerated Vecchia Approximations of Gaussian Processes for Geospatial Data using Batched Matrix Computations

Qilong Pan[1], Sameh Abdulah[2], Marc G. Genton[12], David E. Keyes[2], Hatem Ltaief[2], Ying Sun[12]
Division of Computer, Electrical, and Mathematical Sciences and Engineering (CEMSE),
[1] Statistics Program,
[2] Extreme Computing Research Center,
King Abdullah University of Science and Technology,
Thuwal, Jeddah 23955, Saudi Arabia

***Abstract*—Gaussian processes (GPs) are commonly used for geospatial analysis, but they suffer from high computational complexity when dealing with massive data. For instance, the log-likelihood function required in estimating the statistical model parameters for geospatial data is a computationally intensive procedure that involves computing the inverse of a covariance matrix with size $n \times n$, where n represents the number of geographical locations in the simplest case. As a result, in the literature, studies have shifted towards approximation methods to handle larger values of n effectively while maintaining high accuracy. These methods encompass a range of techniques, including low-rank and sparse approximations. Among these techniques, Vecchia approximation is one of the most promising methods to speed up evaluating the log-likelihood function. This study presents a parallel implementation of the Vecchia approximation technique, utilizing batched matrix computations on contemporary GPUs. The proposed implementation relies on batched linear algebra routines to efficiently execute individual conditional distributions in the Vecchia algorithm. We rely on the KBLAS linear algebra library to perform batched linear algebra operations, reducing the time to solution compared to the state-of-the-art parallel implementation of the likelihood estimation operation in the *ExaGeoStat* software by up to 700X, 833X, 1380X on 32GB GV100, 80GB A100, and 80GB H100 GPUs, respectively, with the largest matrix dimension that can fully fit into the GPU memory in the dense Maximum Likelihood Estimation (MLE) case. We also successfully manage larger problem sizes on a single NVIDIA GPU, accommodating up to 1 million locations with 80GB A100 and H100 GPUs while maintaining the necessary application accuracy. We further assess the accuracy performance of the implemented algorithm, identifying the optimal settings for the Vecchia approximation algorithm to preserve accuracy on two real geospatial datasets: soil moisture data in the Mississippi Basin area and wind speed data in the Middle East.**

***Index Terms*—Gaussian processes (GPs), Vecchia approximation, GPU computing, linear algebra, and batched solvers.**

I. Introduction

Gaussian Processes (GPs) play a crucial role in spatial statistics applications, where they are employed for modeling and predicting geospatial data. This is achieved by defining the mean and covariance functions of the process within a region. For GPs, a parametric form of the covariance function defines the correlation between the spatial locations using a set of parameters, thereby characterizing the dependence of the spatial data. GPs are employed in many applications, such as geostatistics, machine learning, and computer vision. However, a significant challenge arises when dealing with large datasets collected at irregularly spaced locations where the computational complexity of the GP modeling and prediction increases cubically with the number of spatial locations. This poses a limitation for applications with large spatial datasets.

Numerous studies have addressed the computational issues associated with large-scale GP modeling and prediction. Most of the efforts have centered on two main directions: sparse approximation and low-rank approximation to the covariance matrix. Examples of the former include covariance tapering methods [1], [2], [3]. In covariance tapering, the covariance function is multiplied by a tapering function that decays to zero as the distance between two locations increases. This process produces from the original dense covariance method a sparse one that can be managed less cumbersomely. Other studies assume the sparsity of the original covariance matrix by partially including correlation between some spatial locations and ignoring others, i.e., sparse inverse covariance methods [4], [5]. Moreover, the emergence of modern hardware architectures that support low-precision computation, such as NVIDIA GPUs, has facilitated the optimization of sparse inverse covariance methods by applying different precisions to various parts of the dense covariance matrix to reduce the computational complexity instead of ignoring them [6], [7], [8]. For the latter, different types of low-rank approximations are exploited, which allows faster computation and less memory consumption compared to the original dense matrix [9], [10], [11], [12].

Vecchia approximation is one of the earliest GP statistical approximation methods. It involves replacing the high-dimensional joint distribution of the GP with a product of univariate conditional distributions. A small set of observations is conditioned in each conditional distribution, as described in [13]. This method involves a portion of the locations at once instead of all the locations in the modeling process, allowing faster execution and less memory consumption. Vecchia ap-

proximation results in an approximated log-likelihood function with a computational complexity of $\mathcal{O}(nm^3)$ instead of the standard $\mathcal{O}(n^3)$ complexity. Here, n denotes the number of spatial locations, and m represents the number of neighbors considered in the conditional distributions, which is significantly smaller than n. Another advantage of the Vecchia approximation is that it is amenable to parallel computing since terms may be computed independently. However, a challenge to scaling Vecchia approximation is that it requires small matrix operations, which can be more suitable to parallelize on CPUs rather than GPUs. While GPUs are intended to solve problems with huge computational requirements, numerous applications, including Vecchia approximation, offer many small tasks instead. For reviews in Vecchia approximation, see [14], [15], [16], [17].

The latest TOP500 Supercomputers list released in November 2023 reveals that 9 of the top 10 supercomputers worldwide use NVIDIA, Intel, or AMD GPU accelerators, allowing peak performance levels of more than 1.6 ExaFlops/s [18]. GPUs are favored to accelerate tasks because of their superior computational power and energy efficiency compared to CPUs. With CPUs in complex matrix operations, accelerators have traditionally been employed to handle the computation. GPUs are typically leveraged for computationally-intensive tasks, while CPUs are better suited for latency-sensitive ones. However, this approach does not work well with small matrix operations that do not fully utilize the existing accelerators. Instead, concurrent batched operations can be used to execute the same operation across multiple small matrices on a single GPU to allow better exploitation of the existing hardware [19], [20].

In this work, we leverage the power of modern GPU architectures to accelerate the Vecchia approximation algorithm of the Gaussian field using batched matrix operations. The approach applies uniform operations to batches of small matrices to leverage the underlying GPU accelerators. We assess the execution and accuracy performance of our implementation on three different NVIDIA GPU accelerators, GV100, A100, and H100, showing fast execution using batched matrix operations with an accuracy comparable to the dense solution provided with the state-of-the-art Gaussian process software, i.e., *ExaGeoStat* [5]. This study also shows we can handle larger problem sizes with Vecchia approximation on a single GPU compared to using exact likelihood. In our numerical study and real dataset analysis, we find that the suitable number of neighbors (conditioning set) required for modeling each location is at most 60. Vecchia approximation effectively reduces the memory complexity of the MLE operation from $\mathcal{O}(n^2)$ to $\mathcal{O}(nm^2)$, e.g., $m \leq 60$. In the experimental section, we assess the performance and accuracy of our implemented approach, emphasizing the benefits of utilizing Vecchia approximation over the exact likelihood while maintaining the necessary accuracy.

The paper is structured as follows: In Section II, we summarize our contributions. In Section III, we review related work. Section IV provides a comprehensive background for the paper. Section V offers a detailed explanation of our proposed implementation. Section VI presents the evaluation of our implementation from both accuracy and performance perspectives, and we conclude in Section VII.

II. Contributions

We summarize the contributions of the paper as follows:

- We introduce a GPU-accelerated implementation of the well-known Vecchia approximation algorithm for estimating statistical model parameters in the context of climate and weather applications.
- We utilize the KBLAS library and batched linear algebra operations to enhance the speed of our implementation on contemporary GPU architectures, including those from NVIDIA, such as GV100, A100, and H100.
- We assess the accuracy of the proposed implementation through numerical study and two real datasets: a soil moisture dataset from the Mississippi Basin area and a wind speed dataset from the Middle East region. We emphasize identifying the optimal settings that allow the Vecchia algorithm to achieve performance on par with the exact MLE operation as implemented in state-of-the-art HPC geostatistics software, i.e., *ExaGeoStat*.
- We assess the execution performance of the GPU-based Vecchia algorithm on three different NVIDIA GPU architectures, achieving speedups of up to 700X, 833X, 1380X on 32GB GV100, 80GB A100, and 80GB H100 GPUs, respectively, compared to the exact MLE operation.
- Our implementation accommodates larger problem sizes within the same GPU memory, enabling improved modeling for high-resolution geospatial data.

III. Related Work

A. Vecchia Approximation

The Vecchia method, as described in the study on Gaussian process estimation [13], has been investigated and proven to be computationally feasible for non-gridded spatial data. The basic idea behind the Vecchia approximation is to approximate the full covariance matrix of the Gaussian process by considering a smaller subset of the data points and using a conditional independence assumption. This approximation is particularly useful in cases where the full covariance matrix is too large to handle efficiently. The study [21] offers an in-depth examination of the Vecchia method, highlighting its effectiveness in handling spatial data with various characteristics. In [21], different spatial orderings are investigated to enhance the approximation of Gaussian processes via the Vecchia algorithm. Specifically, maximum–minimum distance and random orderings demonstrate a remarkable 99% relative efficiency of the approximation algorithm while requiring only a minimal set of 30 neighboring data points. Furthermore, an idea of grouping locations that share the most common neighbors has been introduced to expedite computations while gaining accuracy. Subsequently, to adapt the Vecchia approximation for MLE operation, the Fisher scoring optimization algorithm [22] has been employed, resulting in MLE convergence within

a few iterations. A dedicated R package, GpGp [22], has also been developed to facilitate these computations. The GpGp team achieved victory in recent competitions assessing the effectiveness of existing software in statistical parameter estimation and prediction, thanks to their utilization of the GpGp package [23], [24], [25].

A general framework for the Vecchia approximation has been introduced [14]. This framework unifies the estimation, prediction, and emulation with the Vecchia approximation while establishing seamless integration with other Gaussian approximation methodologies. The method is primarily deployed to approximate the likelihood of statistical models, particularly in scenarios characterized by expensive computations and complicated modelings. Notable applications include addressing intractable spatial extremes models [26], optimizing Bayesian processes at a large scale [17], and advancing the compositional warpings to construct nonstationary spatio-temporal covariance models [27]. These applications underscore the versatility of the Vecchia method in handling a range of challenging statistical problems.

B. GPU-based Acceleration in Spatial Statistics

GPU accelerators have been employed in spatial statistics to tackle various problems, aiming to efficiently process large-scale data and maximize data utilization for parameter estimation in statistical models. For instance, [28] primarily concentrates on data parallel approaches for tasks such as spatial indexing, spatial joins, polygon rasterization, decomposition, and point interpolation. These approaches are further expanded to encompass distributed computing nodes by integrating multiple GPU implementations.

Another example is [29], which focuses on accelerating the computation of high-order spatial statistics, particularly in geology, by introducing a GPU-based parallel approach, significantly enhancing the computational efficiency of high-order spatial statistics. The key feature of this approach is the utilization of template-stage parallelism. In a real-world application involving a large-scale dataset, [30] tackles the significant computational challenge of handling over 375 million species occurrence records sourced from the global biodiversity information facility by leveraging GPUs. Their research is notable for the impressive performance enhancements attained by applying parallel zonal statistics. Furthermore, [31] presents a novel GPU-accelerated approach for adaptive kernel density estimation to address the computational challenges related to bandwidth determination in spatial point pattern analysis. This method incorporates optimizations that reduce algorithmic complexity and harness GPU parallelism, resulting in significant speed improvements. [32] also underscores the importance of expediting geospatial computations and analytics through shared and distributed memory parallel platforms. They advocate using GPUs, which boast hundreds to thousands of processing cores for parallelization. When working with spatial Gaussian data, these datasets are typically represented as realizations derived from a Gaussian spatial random field. The GPU-based parallel solutions presented in their work underscore the promising potential of GPU technology in this field.

IV. Background

A. Gaussian Process and Likelihood Function

Statistical modeling involves analyzing correlations among various spatial or spatio-temporal points distributed regularly or irregularly over a geographical area. In the purely spatial scenario, these datasets are assumed to be realizations observed from a Gaussian spatial random field. Consider a set of n locations, $\mathbf{s}_1, \ldots, \mathbf{s}_n \in \mathbb{R}^d$, and their observations, $\mathbf{y} = (y_1, \ldots, y_n)^\top$ which is indicated by $y_i := y(\mathbf{s}_i) \in \mathbb{R}$. We model the data $\mathbf{y}$ with Gaussian process, $\mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma}_{\boldsymbol{\theta}})$, where $\boldsymbol{\Sigma}_{\boldsymbol{\theta}}$ is a covariance matrix with (i, j) entry determined by a given covariance function $C_{\boldsymbol{\theta}}(\mathbf{s}_i, \mathbf{s}_j)$ relying on a vector of covariance parameters, $\boldsymbol{\theta}$. Without loss of generality, we assume that $\mathbf{y}$ has a mean of zero. Statistical inference about $\boldsymbol{\theta}$ is often based on the Gaussian log-likelihood function, which entails a cubic computational complexity [5]:

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

The maximum likelihood estimator of $\boldsymbol{\theta}$ is the value $\hat{\boldsymbol{\theta}}$ that maximizes $\ell(\boldsymbol{\theta})$ in the equation (1). Examples of existing covariance functions are the isotropic Matérn covariance function [33] in (2) and the power exponential kernel in (3):

$$C(\mathbf{s}_i, \mathbf{s}_j) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\frac{\|\mathbf{s}_i - \mathbf{s}_j\|}{\beta} \right)^\nu K_\nu \left(\frac{\|\mathbf{s}_i - \mathbf{s}_j\|}{\beta} \right), \quad (2)$$

$$C(\mathbf{s}_i, \mathbf{s}_j) = \sigma^2 \exp\left(-\frac{\|\mathbf{s}_i - \mathbf{s}_j\|^\nu}{\beta} \right), \quad (3)$$

where $\boldsymbol{\theta} = \left(\sigma^2, \beta, \nu\right)^\top$, σ^2 is the variance, $\Gamma(\cdot)$ is the gamma function, $K_\nu(\cdot)$ is the Bessel function of the second kind of order ν, and $\beta > 0$ and $\nu > 0$ are range and smoothness parameters, respectively.

B. Kullback-Leibler (KL) Divergence

KL divergence is a statistical metric to measure the difference between two given probability distributions. Assuming two models P and Q, the KL divergence of P from Q is the amount of information lost when using Q as a model compared to the actual distribution P. It is represented as $D_{\mathrm{KL}}(P \parallel Q)$, with P and Q representing the two probability distributions under comparison. KL divergence is used in various fields, including information theory, machine learning, and statistics.

For two distributions, P and Q of a continuous random variable, KL divergence is defined as,

$$D_{\mathrm{KL}}(P \parallel Q) = \int_{-\infty}^{\infty} p(x) \log\left(\frac{p(x)}{q(x)} \right) dx,$$

where p and q denote the probability densities of P and Q. In the Gaussian fields, $P = \mathcal{N}_0(\mathbf{0}, \boldsymbol{\Sigma}_0)$ and $Q = \mathcal{N}_1(\mathbf{0}, \boldsymbol{\Sigma}_1)$, the KL divergence of $\mathcal{N}_0$ and $\mathcal{N}_1$ is [34]:

$$D_{\mathrm{KL}}(\mathcal{N}_0 \parallel \mathcal{N}_1) = \frac{1}{2}\left\{ \mathrm{tr}\left(\boldsymbol{\Sigma}_1^{-1}\boldsymbol{\Sigma}_0\right) - k + \log\frac{|\boldsymbol{\Sigma}_1|}{|\boldsymbol{\Sigma}_0|} \right\}, \quad (4)$$

where $\mathbf{\Sigma}_0 := \mathbf{\Sigma}(\boldsymbol{\theta}_0)$ and $\mathbf{\Sigma}_1 := \mathbf{\Sigma}(\boldsymbol{\theta}_1)$.

In this paper, we use the KL divergence criterion to assess the loss of information when relying on the Vecchia approximation algorithm compared to the exact Gaussian likelihood. Thus, with the plug-in of the Vecchia approximated Gaussian likelihood in (4), $Q = \mathcal{N}_a(\mathbf{0}, \mathbf{\Sigma}_a)$ where $\mathbf{\Sigma}_a$ represents an approximated covariance matrix to $\mathbf{\Sigma}_0$, (4) is simplified as,

$$D_{\mathrm{KL}}(\mathcal{N}_0 \parallel \mathcal{N}_a) = \ell_0(\boldsymbol{\theta}; \mathbf{0}) - \ell_a(\boldsymbol{\theta}; \mathbf{0}), \tag{5}$$

where $\ell_0(\boldsymbol{\theta}; \mathbf{0})$ is the exact log-likelihood at the $\mathbf{y} = \mathbf{0}$ while $\ell_a(\boldsymbol{\theta}; \mathbf{0})$ is the Vecchia-approximated log-likelihood at $\mathbf{y} = \mathbf{0}$.

C. Batched Linear Algebra Computations

Considering the case of block-sparse matrices, non-zero elements are clustered in different parts of the matrix, providing an alternative approach to handling them compared to using dense or sparse linear solvers. The size of these dense blocks may suggest treating them as a collection of small, dense matrices instead. In the literature, performing identical operations on multiple small dense matrices is known as batched execution. Therefore, the concept of batched linear algebra routines is often employed in various applications where the same operations can be applied simultaneously to multiple small, dense matrices. These operations are performed independently on each matrix, enabling more efficient use of computational resources, particularly on high-performance hardware like GPUs.

Numerous software libraries have facilitated linear algebra operations in dense and approximate formats. These include LAPACK [35], SLATE [36], PLASMA [37], HiCMA [38], H2Opus [39], MAGMA [40], KBLAS [41], and BLIS [42]. Several of these libraries offer methods to enhance efficient batched processing on both CPUs and GPUs, including batched routines for various Basic Linear Algebra Subprograms (BLAS) and LAPACK operations. The Batched BLAS extension enhances the traditional BLAS library by enabling simultaneous processing of multiple small matrices (up to 1024). Its main goal is to diminish the computational burden caused by frequent function calls. Designed for modern hardware, it optimally utilizes the parallel processing capabilities of multi-core processors and GPUs [43], [27].

TABLE I: The impact of three data layouts

	Pros	Cons
P2P	Flexible appendage	Noncontiguous data
Strided	Contiguous data	Expensive appendage
Interleave	Vectorization	Complexity format

In this context, our focus is directed toward GPU-batched operations. Presently, three commonly used libraries for conducting batched operations are cuBLAS, MAGMA [43], and KBLAS [41]. It is essential to note that each of the three libraries offers distinct data layouts and functions. The functions are essential to the Vecchia algorithm, and the data layout greatly impacts the performance. The distinction becomes especially significant when considering the choice of data layout, which can be pointer-to-pointer (P2P), strided, or interleaved formats, as shown in Table I. The P2P data layout offers the flexibility to append additional data at the end of the batched data array by adding a pointer. However, this convenience is offset by the memory loading burden stemming from the non-contiguous nature of the small matrices, which are scattered across the global memory on the GPU. Conversely, the strided data layout allocates contiguous memory for the entire collection of small matrices, thus resulting in efficient access to these matrices. Nevertheless, adding extra data incurs significant expense in terms of memory usage. On the other hand, the interleaved layout is distinguished by its ability to utilize memory through vectorization fully. However, this benefit is primarily realized for very small matrices (less than 24) [44].

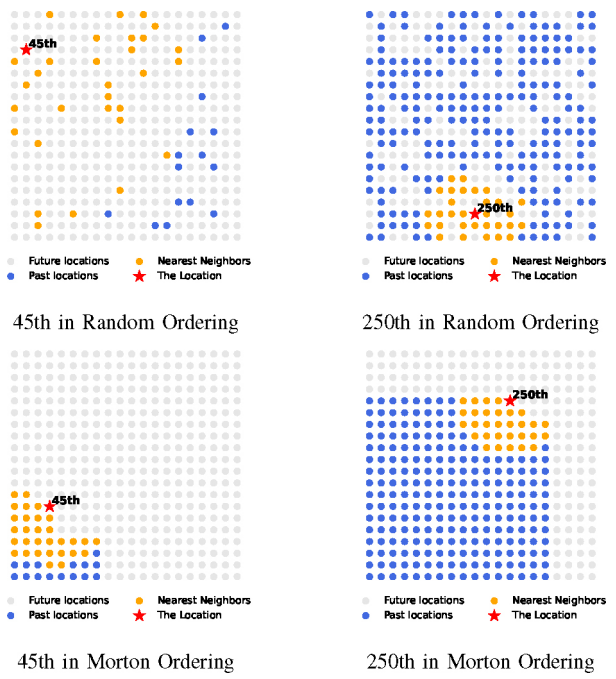
V. Batched Vecchia Approximation

This section offers an in-depth description of our proposed framework for the batched Vecchia algorithm. It commences by explaining the preprocessing step, which entails reordering the location set and selecting the nearest neighbors for each location from previous locations. Subsequently, we delve into the memory requirements for the batched Vecchia algorithm with a comprehensive description of the proposed implementation. Finally, we compare the expected memory usage and computational complexity of the Vecchia algorithm in contrast to the exact MLE solution.

A. Reorderings

The initial step of the Vecchia algorithm involves efficiently reordering the locations to identify nearest-neighbor points for each location. Consequently, selecting the reordering method plays a crucial role in ensuring the accuracy of the log-likelihood approximation. This is because, for a given observation, the selection of nearest neighbors from prior observations highly depends on the sequence in which the data is arranged. According to [22], the maximum–minimum distance (Maxmin) ordering method is identified as suitable for the Vecchia algorithm. Notably, random ordering achieves comparable accuracy to Maxmin, as demonstrated in [21]. However, when dealing with large-scale problems, employing the Maxmin ordering method can become impractical due to its computational complexity, which can scale as $\mathcal{O}(n^3)$. Furthermore, the GpGp R package recommends random ordering for scenarios where locations exceed 100K [22]. Consequently, random ordering has been used in our experiments. In addition to random ordering, the Morton ordering method is also considered, given its efficiency in tile-low rank Cholesky factorization as reported in [45].

To elucidate the impacts of two distinct ordering methods on finding the best nearest neighbors, we present an illustrative example involving a grid of 20×20 locations, as depicted in Figure 1. This example demonstrates that, in contrast to the Morton ordering method, random ordering confers an advantage for locations sequenced toward the end of the ordering process. It retains the nearest neighbors immediately



45th in Random Ordering | 250th in Random Ordering

45th in Morton Ordering | 250th in Morton Ordering

Fig. 1: The example of random and Morton ordering on locations 20×20. (First row) The 45th and 250th locations (red stars) in the random ordering are marked with their nearest neighbors (orange circle); (Second row) The 45th and 250th locations (red stars) in the Morton order algorithm are marked with their nearest neighbors (orange circle). Blue circles indicate past locations for a given ordering algorithm.

surrounding a target location. However, this comes initially at the expense of losing proximity accuracy for locations ordered. In the experimental section, we will apply both methods to reorder our locations and empirically demonstrate which can yield superior results.

B. Batched Vecchia Approximated Likelihood Algorithm

In an analysis involving a set of geospatial data points $\mathbf{y}$ across various locations, and their observed values, when we apply any reordering τ to the sequence of these locations, the likelihood expression for the dataset retains its form when represented as a sequence of conditional densities,

$$L(\boldsymbol{\theta}; \mathbf{y}) = p_{\boldsymbol{\theta}}\left(y_1, \ldots, y_n\right) \tag{6}$$

$$= p_{\boldsymbol{\theta}}\left(y_1^{\tau}\right) \prod_{i=2}^{n} p_{\boldsymbol{\theta}}\left(y_i^{\tau} \mid y_1^{\tau}, \ldots, y_{i-1}^{\tau}\right). \tag{7}$$

Vecchia approximation is expressed as (9) and it replaces the conditioning vectors $\left(y_1^{\tau}, \ldots, y_{i-1}^{\tau}\right)$ with $\left(y_{j_{i1}}^{\tau}, \ldots, y_{j_{im_i}}^{\tau}\right)$ which are nearest neighbors to the target $y_{j_i}^{\tau}$; J_i is defined as $\{j_{i1}, \ldots, j_{im_i}\}$ and $J = \{J_1, \ldots, J_n\}$. [21]

$$p_{\boldsymbol{\theta},\tau,J}\left(y_1, \ldots, y_n\right) = p_{\boldsymbol{\theta}}\left(y_1^{\tau}\right) \prod_{i=2}^{n} p_{\boldsymbol{\theta}}\left(y_i^{\tau} \mid y_{j_{i1}}^{\tau}, \ldots, y_{j_{im_i}}^{\tau}\right) \tag{8}$$

$$= p_{\boldsymbol{\theta}}\left(y_1^{\tau}\right) \prod_{i=2}^{n} p_{\boldsymbol{\theta}}\left(y_i^{\tau} \mid \mathbf{y}_{J_i}^{\tau}\right). \tag{9}$$

The accuracy of the Vecchia algorithm's approximation is dependent on the choice of permutation τ and J, because the permutation τ creates specific ordering from 2D to 1D and the J represents the deviations from the exact likelihood. [21] [13]

To implement the Vecchia algorithm, for each spatial location, we must compute a covariance matrix of its nearest neighbors and a cross-covariance vector between the location and its nearest neighbors. Figure 2 depicts the required scalar/vector/matrix for each spatial location, while Figure 3 shows the contiguous data allocation utilized in the hardware. y_i^{τ} and $\mathbf{y}_{J_i}^{\tau}$ (orange) are the ith observation and its neighbors' observations which exactly match the representation in equation (9); σ_i and $\boldsymbol{\Sigma}_i$ (green) are the corresponding variance and covariance matrix for the ith observation and its neighbors; $\mathbf{v}_i$ (green) is the cross-covariance matrix of y_i^{τ} and $\mathbf{y}_{J_i}^{\tau}$. For every pair $(y_i^{\tau}, \mathbf{y}_{J_i}^{\tau}, \sigma_i, \boldsymbol{\Sigma}_i)$, their log-likelihoods, $p_{\boldsymbol{\theta}}\left(y_i^{\tau} \mid \mathbf{y}_{J_i}^{\tau}\right)$, are independent of each other. That is to say, the task of computing log-likelihood in the (9) can be divided into n independent tasks. In the log-likelihood computation, the batched operations are conducted with regard to $\mathbf{v}_i$, $\boldsymbol{\Sigma}_i$ (green), $\mathbf{y}_{J_i}^{\tau}$ (orange) where the two dummy dashed vectors are not used for log-likelihood calculation but purely for simplifying the batched algorithm.

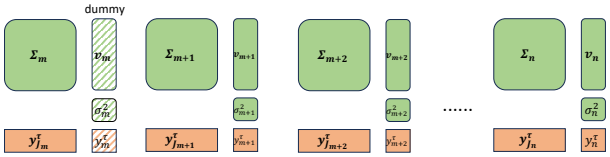


Fig. 2: Batched Vecchia algorithm description. $\boldsymbol{\Sigma}_{m:n}$ are constructed by the nearest neighbors of $\mathbf{y}_{m:n}^{\tau}$. The batched POTRF routine is applied to these matrices. After this decomposition, the resulting outputs are utilized as inputs for the batched TRSV operation with $\mathbf{v}_{m:n}$ and $\mathbf{y}_{\mathbf{J}_{m:n}}^{\tau}$, separately.

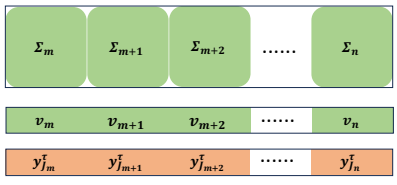


Fig. 3: Contiguous data allocation in the GPU global memory.

Algorithm 1 provides a high-level overview of the batched operations employed in the Vecchia approximation, as depicted

Algorithm 1 Batched Vecchia algorithm

1: **Input:** m, n, τ, $C(\cdot,\cdot)$
2: **Output:** ℓ (log-likelihood)
3: $\tau : \{(\mathbf{s}_1, y_1), \ldots, (\mathbf{s}_n, y_n)\} \rightarrow \{(\mathbf{s}_1^\tau, y_1^\tau), \ldots, (\mathbf{s}_n^\tau, y_n^\tau)\}$ ▷ Permutation
4: **while** $m+1 \leq j \leq n$ **do** ▷ Nearest neighbors
5: $\quad J_j \leftarrow mNearstNeighbors(\mathbf{s}_1^\tau, \mathbf{s}_2^\tau, \ldots, \mathbf{s}_{j-1}^\tau; \mathbf{s}_j^\tau, m)$
6: **end while**
7:
8: $\boldsymbol{\sigma}_{m:n} \leftarrow batchC(\mathbf{s}_{m:n}^\tau, \mathbf{s}_{m:n}^\tau)$ ▷ Batched Kernel
9: $\boldsymbol{\Sigma}_{m:n} \leftarrow batchC(\mathbf{s}_{J_{m:n}}^\tau, \mathbf{s}_{J_{m:n}}^\tau)$
10: $\mathbf{v}_{m:n} \leftarrow batchC(\mathbf{s}_{m:n}^\tau, \mathbf{s}_{J_{m:n}}^\tau)$
11: $\mathbf{y}_{J_{m:n}}^\tau$
12: $\boldsymbol{\sigma}^{old} \leftarrow (\sigma_m, \ldots, \sigma_n)^T$
13: $\boldsymbol{L}_{m:n} \leftarrow batchPOTRF(\boldsymbol{\Sigma}_{m:n})$ ▷ Batched operations
14: $\mathbf{v}'_{m:n} \leftarrow batchTRSV(\boldsymbol{L}_{m:n}, \mathbf{v}_{m:n})$
15: $\mathbf{y}'^\tau_{j_{m:n}} \leftarrow batchTRSV(\boldsymbol{L}_{m:n}, \mathbf{y}^\tau_{j_{m:n}})$
16: $\boldsymbol{Y}' = (\mathbf{y}_{j_m}^\tau, \mathbf{y}_{j_{m+1}}^\tau, \ldots, \mathbf{y}_{j_n}^\tau)$ ▷ Concatenate
17: $\boldsymbol{V}' = (\mathbf{v}_m, \mathbf{v}_{m+1}, \ldots, \mathbf{v}_n)$
18: $\mathbf{v}_m \leftarrow \mathbf{y}_{j_m}^\tau$
19: $\boldsymbol{\mu}' \leftarrow DotProduct(\boldsymbol{Y}'^T, \boldsymbol{V}')$ ▷ Correction vectors
20: $\boldsymbol{\sigma}' \leftarrow DotProduct(\boldsymbol{V}'^T, \boldsymbol{V}')$
21: $\ell_m \leftarrow -computeLogDet(\boldsymbol{L}_m) - \frac{\mu'_m}{2} - \frac{m}{2} log(2\pi)$
22: $\boldsymbol{\mu}^{new} \leftarrow \boldsymbol{\mu}'_{(m+1):n}$ ▷ Vecchia updates
23: $\boldsymbol{\sigma}^{new} \leftarrow \boldsymbol{\sigma}^{old} - \boldsymbol{\sigma}'_{(m+1):n}$
24: **while** $(m+1) \leq i \leq n$ **do** ▷ Univariate Gaussian ℓ
25: $\quad \ell_i \leftarrow -\frac{1}{2}\left(\left(\frac{x-\mu_i^{new}}{\sigma_i^{new}}\right)^2 + log(2\pi) + 2log(\sigma_i^{new})\right)$
26: **end while**
27: $\ell \leftarrow \ell_m + \ell_{(m+1)} + \ldots + \ell_n$

in Figure 2. The inputs are m, representing the size of the conditioning set, n, the total number of observations, τ, the permutation set of the locations, and $C(\cdot,\cdot)$, the specified covariance function. The output is the log-likelihood estimation ℓ. The following points outline the specific low-level implementation details:

- *GPU acceleration:* In Algorithm 1, lines 9 and 10 describe the utilization of a batched CUDA kernel to generate the covariance matrix [46], which overcomes the primary computational bottleneck in Vecchia approximation as discussed in [21]. Following this, we leverage batched operations, notably the Cholesky decomposition for symmetric matrices (POTRF) and the subsequent use of a triangular linear solver (TRSV), as demonstrated in lines 13, 14, and 15, using the KBLAS library [41]. In lines 19 and 20, we introduce a CUDA kernel implementation for dot product computation. In this context, each thread calculates an individual vector. This approach is particularly efficient in the Vecchia framework, where vector sizes typically range from 30 to 120 elements, but the quantity of vectors extends to 500K or more. Consequently, allocating a single thread for each dot product operation is more effective than batched processing.
- *Data layout:* The KBLAS library [41] provides two data layouts: Point-to-Point (P2P) and strided. The strided layout is especially beneficial for the Vecchia algorithm for several reasons. First, it eliminates the need to merge previous data with new data points since all spatial or temporal data and observations are preloaded, avoiding the necessity for streaming access. Second, the strided layout ensures continuous data storage, enhancing GPU memory use efficiency. This is particularly important for managing the large-scale problems typical of the Vecchia algorithm. The data structure is shown in the Figure 3.
- *Memory Allocation:* The main memory allocation challenge in our algorithm arises from the large number of dense, small matrices. The space complexity for this part of the algorithm is $\mathcal{O}(nm^2)$, where m is the size of each small matrix, and n is the total number of matrices. Besides, memory allocation for small vectors has space complexity of $\mathcal{O}(nm)$, which is particularly less impactful in the overall memory usage compared to the contiguous small matrices. These contiguous matrices and vectors are stored in the global memory of GPU.

C. Memory Footprint and Arithmetic Complexity of Batched Vecchia

In this subsection, we analyze the memory footprint and the arithmetic complexity of the Batched Vecchia implementation proposed in *ExaGeoStat*, comparing it to the exact MLE implementation. The memory footprint of the exact MLE algorithm is $\sim n^2/2$ for the symmetric covariance matrix $\boldsymbol{\Sigma}$ and $\sim n$ for the measurements vector $\boldsymbol{y}$. For the Vecchia algorithm, each location requires a covariance matrix $\boldsymbol{\Sigma}_i$, a vector $\boldsymbol{v}_i$, and a measurement vector $\boldsymbol{y}$, with memory complexities of $\sim n(m^2/2)$, $\sim nm$, and $\sim nm$, respectively. Figure 4 (a) illustrates memory footprint in gigabytes (GB) for various problem sizes when employing the Vecchia and exact MLE algorithms. The figure emphatically highlights the benefits of utilizing the Vecchia approximation, which exhibits significantly lower memory requirements. The values 30, 60, 90, 120, and 150 represent the number of nearest neighbors considered for each location, which is represented by m.

The Vecchia algorithm significantly reduces the computational complexity compared to the exact MLE. For exact MLE, the complexity primarily stems from the Cholesky factorization of the covariance matrix, $\sim n^3/3$, and the triangular solve, $\sim n^2$. In contrast, with the Vecchia approximation, the complexity for the Cholesky factorization operations is $\sim n(m^3/3)$ (as shown in line 14 in Algorithm 1), and there are triangular solves with vector $\boldsymbol{v}_i$ (line 15) and triangular solves with vector $\boldsymbol{y}_i$ (line 16). Figure 4 (b) demonstrates the floating-point operations (flops) in Gflops for various problem sizes when using both the Vecchia and exact MLE algorithms. The figure shows a significant reduction in the number of required flops for the Vecchia algorithm, underlining its efficiency, particularly if it maintains accuracy comparable to the exact MLE solution.

In these two aforementioned subfigures, we also show the arithmetic complexity of large conditioning sets m for the batched Vecchia that exceeds the memory requirement of the exact MLE. Figure 4 (a) shows that when $m = 1200$, the memory requirement of the batched Vecchia algorithm exceeds the exact MLE algorithm with different problem sizes. However, Figure 4 (b) shows that Vecchia still requires fewer flops than exact MLE at the same conditioning set size.

VI. Results and Discussions

In this section, we conduct a series of experiments to evaluate the accuracy and performance of the batched Vecchia algorithm with several goals: (1) Numerically assess the accuracy of the batched Vecchia algorithm by comparing it to the exact MLE using KL divergence. (2) Evaluate the accuracy of the batched Vecchia algorithm using real datasets, focusing on modeling and prediction accuracy. (3) Examine the performance of the implemented algorithm across various NVIDIA GPUs, specifically the GV100, A100, and H100. (4) Investigate the largest problem size manageable with different conditioning sets in the Vecchia algorithm on various GPUs. (5) Discuss optimal parameters for the batched Vecchia approximation to achieve the best performance while maintaining the necessary accuracy.

A. Experimental Testbed

We conduct accuracy and performance assessment experiments using a range of GPUs, including a single NVIDIA GV100 with 32 GB of memory, a single NVIDIA A100 with 80 GB of memory, and an H100 with 80 GB of memory.

Our computational harness is built using gcc version 10.2.0 (12.2.0) and CUDA version 11.4 (11.8). It was linked with the KBLAS library, Intel MKL 2022.2.1, MAGMA 2.6.0, and NLopt v2.7.1 optimization libraries. All computations were performed in double-precision arithmetic, and we conducted each experimental run five times to verify repeatability. To assess accuracy and perform qualitative analysis, we utilized numerical calculations and examined two real datasets: the soil moisture dataset from the Mississippi River Basin region and the wind speed dataset from the Middle East region.

B. Numerical Study

In this subsection, we utilize exact log-likelihood calculated by *ExaGeoStat* [11], focusing on Gaussian random fields with problem sizes of 180K and 260K in two-dimensional (2D) spatial locations. We use the Matérn kernel as described in (2) and vary the smoothness parameter ν at values of 0.5, 1.5, and 2.5, corresponding to low, medium, and high smoothness levels. Additionally, for each level of smoothness, we adjust the effective range to 0.1, 0.3, and 0.8 to account for low, medium, and high dependence values, respectively, which impacts data correlation. Detailed configurations can be found in Table II as part of our study on Gaussian random fields.

The calculation of the KL divergence is performed using (5), and the outcomes are visualized in Figure 5 and Figure 6. Across all subfigures, the x-axis corresponds to the size of the conditioning set, while the y-axis represents the KL divergence value, with 0 indicating a perfect match with the exact MLE operation. Examination of these figures yields several significant observations:

TABLE II: The cross combinations of low/medium/high smoothness and low/medium/high effective range. Each entry in the table represents β, and the $0.1, 0.3, 0.8$ are the statistically effective range, i.e., the distance over which spatial dependencies are significant in the statistical model [23].

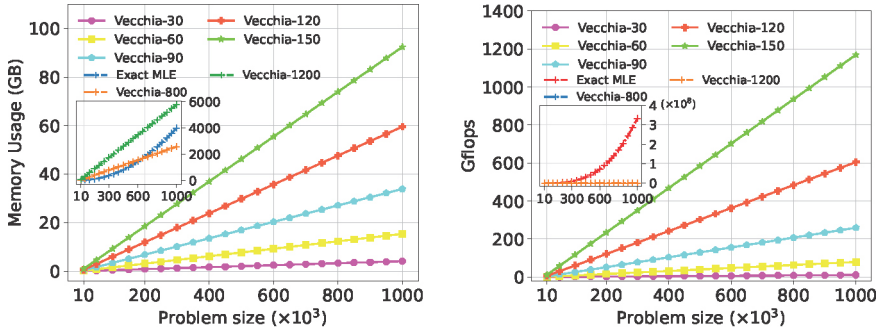
	$\nu = 0.5$	$\nu = 1.5$	$\nu = 2.5$
effective range=0.1	0.026270	0.017512	0.014290
effective range=0.3	0.078809	0.052537	0.014290
effective range=0.8	0.210158	0.140098	0.114318

- The significance of spatial ordering in log-likelihood approximation cannot be overstated. When it comes to accuracy, it becomes evident that random ordering outperforms Morton's ordering at large-scale problems. This observation highlights the role of selecting the right ordering strategy for achieving effective approximations.
- Impact of range and smoothness on approximation difficulty. The complexity of the approximation escalates with increases in range or smoothness parameters. A notable enlarged KL divergence under elevated range or smoothness conditions evidences this. Consequently, the additional conditioning points become imperative in the high-range or high-smoothness scenarios.
- Approximation challenge for large-scale problem size. The difficulty of approximation is directly proportional to the size of the problem. The numerical study reveals an increase in KL divergence when the problem size escalates from 180K to 260K. This observation suggests that larger problem sizes necessitate the use of more conditioning points to maintain approximation accuracy.

C. Accuracy Assessment of Batched Vecchia with Real Data

1) Soil Moisture: This study examines high-resolution daily soil moisture data obtained from the Mississippi River basin in the United States on January 1, 2004, as reported in [47]. The dataset, which was previously utilized in [10] and [5], involves Gaussian field modeling and contains 2 million irregularly distributed locations. To manage computational costs, we randomly selected 250K locations as the training dataset and 25K as the testing dataset. This choice allows us to compare the estimated parameter vector and predictions obtained via the Vecchia approximation with those from exact modeling, where using all 2 million locations would be computationally burdensome. The data has a spatial resolution of 0.0083 degrees, with each one-degree difference approximately corresponding to a distance of 87.5 km. Consequently, we utilize the Great Circle Distance (GCD) metric to calculate the distances between any pair of locations based on their original longitude and latitude values, as described in [5]:

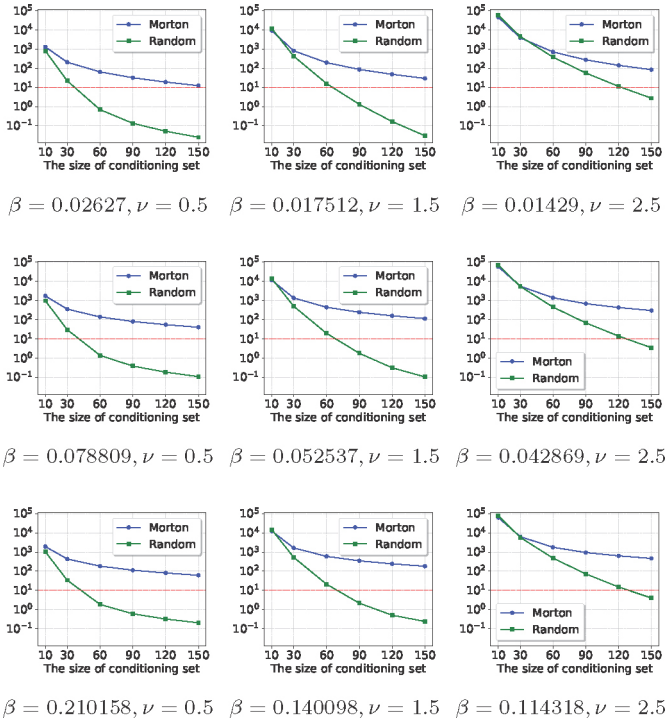
$$\left(\frac{d}{r}\right) = \operatorname{hav}\left(\varphi_2 - \varphi_1\right) + \cos\left(\varphi_1\right)\cos\left(\varphi_2\right)\operatorname{hav}\left(\lambda_2 - \lambda_1\right).$$



(a) Memory complexity.

(b) Computational complexity.

Fig. 4: Comparison of Arithmetic complexity: Vecchia algorithm versus Exact MLE.

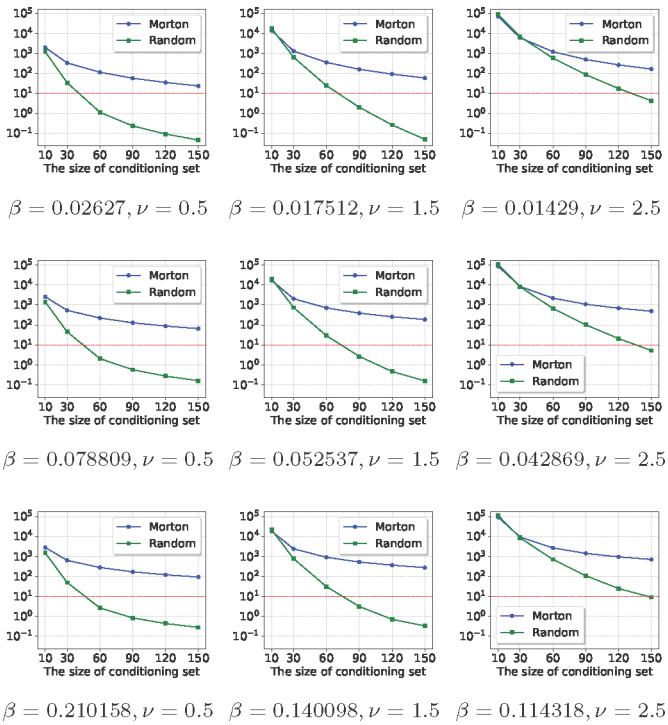


$\beta = 0.02627, \nu = 0.5$ $\beta = 0.017512, \nu = 1.5$ $\beta = 0.01429, \nu = 2.5$

$\beta = 0.078809, \nu = 0.5$ $\beta = 0.052537, \nu = 1.5$ $\beta = 0.042869, \nu = 2.5$

$\beta = 0.210158, \nu = 0.5$ $\beta = 0.140098, \nu = 1.5$ $\beta = 0.114318, \nu = 2.5$

Fig. 5: KL divergence under 180K locations with log10 scale (y-axis is KL divergence). The red dashed line is the recommended threshold for choosing the conditioning size.



$\beta = 0.02627, \nu = 0.5$ $\beta = 0.017512, \nu = 1.5$ $\beta = 0.01429, \nu = 2.5$

$\beta = 0.078809, \nu = 0.5$ $\beta = 0.052537, \nu = 1.5$ $\beta = 0.042869, \nu = 2.5$

$\beta = 0.210158, \nu = 0.5$ $\beta = 0.140098, \nu = 1.5$ $\beta = 0.114318, \nu = 2.5$

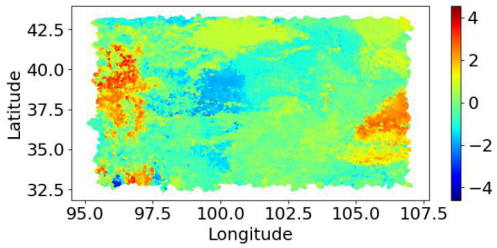
Fig. 6: KL divergence under 260k locations with log10 scale (y-axis is KL divergence). The red dash line is the recommended threshold for choosing the conditioning size.

Here, the haversine function denoted as $\mathrm{hav}(\cdot)$, is defined as $\sin^2\left(\frac{\cdot}{2}\right) = \frac{1-\cos(\cdot)}{2}$, where d represents the distance between two locations, r is the radius of the sphere, φ_1 and φ_2 are the latitudes in radians of locations 1 and 2, respectively, and λ_1 and λ_2 are their respective longitudes.

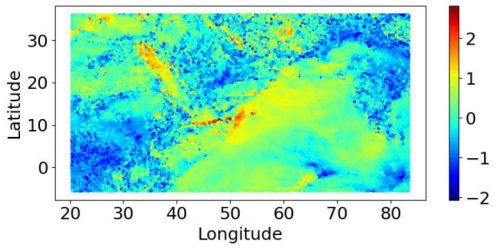
The mean value of the raw dataset is removed by a linear regression model where the response variable is observation, and explanatory variables are longitude and latitude. The residuals, as shown in Figure 7, are fitted using a zero-mean Gaussian process model, which incorporates a power exponential covariance function,

$$C(d) = \sigma^2 \exp\{-d^\alpha/\beta\},$$

where d is the distance between two locations and the parameter vector $\boldsymbol{\theta} = (\sigma^2, \beta, \alpha)^\top$ represent the variance, range and smoothness, respectively. For the Vecchia approximated Gaussian process, six different conditioning sizes (10, 30, 60, 90, 120, 150) with random ordering and 250K subsampling problem size are considered. We use *ExaGeostat* [5] to estimate the parameter vectors for the exact Gaussian process. In



Residual of Observations (soil)



Residual of Observations (wind speed)

Fig. 7: Real datasets residuals: soil moisture and wind speed, where the observations of soil moisture and the square root of wind speed served as the response variables, respectively.

the end, the estimated parameter vectors are plugged into the kriging, and then we obtain the Mean Square Error (MSE) for the prediction task.

2) Wind Speed: The WRF-ARW (Weather Research and Forecasting - Advanced Research WRF) model generated a regional climate dataset specific to the Arabian Peninsula in the Middle East, as documented in [48]. The model is configured with a horizontal grid spacing of 5 km, encompassing 51 vertical levels, and the highest level of the model is established at 10 hPa. Geographically, the model's domain covers the area from 20°E to 83°E in longitude and from 5°S to 36°N in latitude. This dataset spans 37 years, with daily data provided. Each data file contains a complete day's (24 hours) record of hourly wind speed measurements across 17 distinct atmospheric layers. For this study, we focus on the dataset from September 1, 2017, starting at 00:00 AM. Our interest is in wind speed measurements at a height of 10 meters above the ground, corresponding to the lowest layer, referred to as layer 0. The method of calculating distances in the wind speed dataset is consistent with that used in the soil moisture dataset. The residuals of wind speed are visualized in Figure 7, where we utilized the square root of wind speed as the response variable and longitude and latitude as explanatory variables, taking into account the skewed distribution of wind speed—additionally, the same experimental settings as the soil moisture dataset were applied. Herein, the initial dataset comprises 1M locations, from which we randomly extracted 250K locations for training and 25K for testing.

3) Result Analysis: Figure 8 shows the estimated parameter vectors for both datasets, while Table III highlights the MSE

TABLE III: MSE of *ExaGeoStat* and Vecchia on soil and wind dataset ($\times 10^{-2}$).

	ExaGeoStat (Exact MLE)	Vecchia 60	90	120	150
Soil	7.832727	7.850310	7.849056	7.850950	7.850904
Wind	2.842985	2.842939	2.842972	2.842967	2.842954

associated with the prediction. During the estimation process, it was observed that the parameter vector $\boldsymbol{\theta}$, as estimated through the Vecchia approximation, closely aligns with that obtained via *ExaGeoStat* (exact MLE), particularly as the number of conditioning neighbors increases. Figure 8 illustrates that, for both datasets, a conditioning size of 60 is optimal for achieving an estimation close to the exact MLE. Table III further demonstrates that the Vecchia approximation achieves a prediction error remarkably close to the actual values when utilized for predicting missing data.

D. Performance Assessment

In this study, we evaluate the performance of batched Vecchia approximation across three distinct GPU architectures: GV100 (Quadro Volta) with 32 GB, A100(Ampere) with 80 GB, and H100 (Hopper) with 80 GB. The results presented in this subsection represent the average of five separate runs. Our objective is to comprehensively understand the efficacy of our software under varying computational conditions. To this end, we employ a range of conditioning sizes (batch sizes), specifically $(10, 30, 60, 90, 120, 150)$. The selection of smaller sizes is informed by the recommendation of approximately 30 neighbors for optimal Vecchia approximation, as suggested by [21]. Larger values are chosen in response to the demands of extensive problem sizes, which may necessitate increased conditioning sizes. Additionally, we adjust the problem size for each GPU to leverage their memory capacities fully. This approach allows us to explore the upper limits of computational efficiency within the constraints of each hardware configuration. The analysis includes assessing the batched Vecchia method against *ExaGeoStat*-GPU. This comparison focuses on two primary metrics: time and Gflops/sec. Time encompasses the total duration required for computing the log-likelihood, which includes the generation of the covariance matrix and associated log-likelihood operations (POTRF/TRSM/dot product). The Gflops/sec metric specifically evaluates the efficiency of log-likelihood-related operations. The results are depicted in Figure 9, from which several significant insights are discerned:

- *Linear relationship between time and problem size:* Within the same hardware, we observe a linear escalation in time correlating with increases in problem size. This finding provides a useful framework for predicting computational time across varying problem scales.
- *Comparison with ExaGeoStat-GPU:* When evaluated on the metric of a single likelihood estimation time, the Vecchia method with 60 neighbors outperforms *ExaGeoStat*-GPU, exhibiting an approximately 700X, 833X, 1380X speedups compared to exact MLE, where the problem

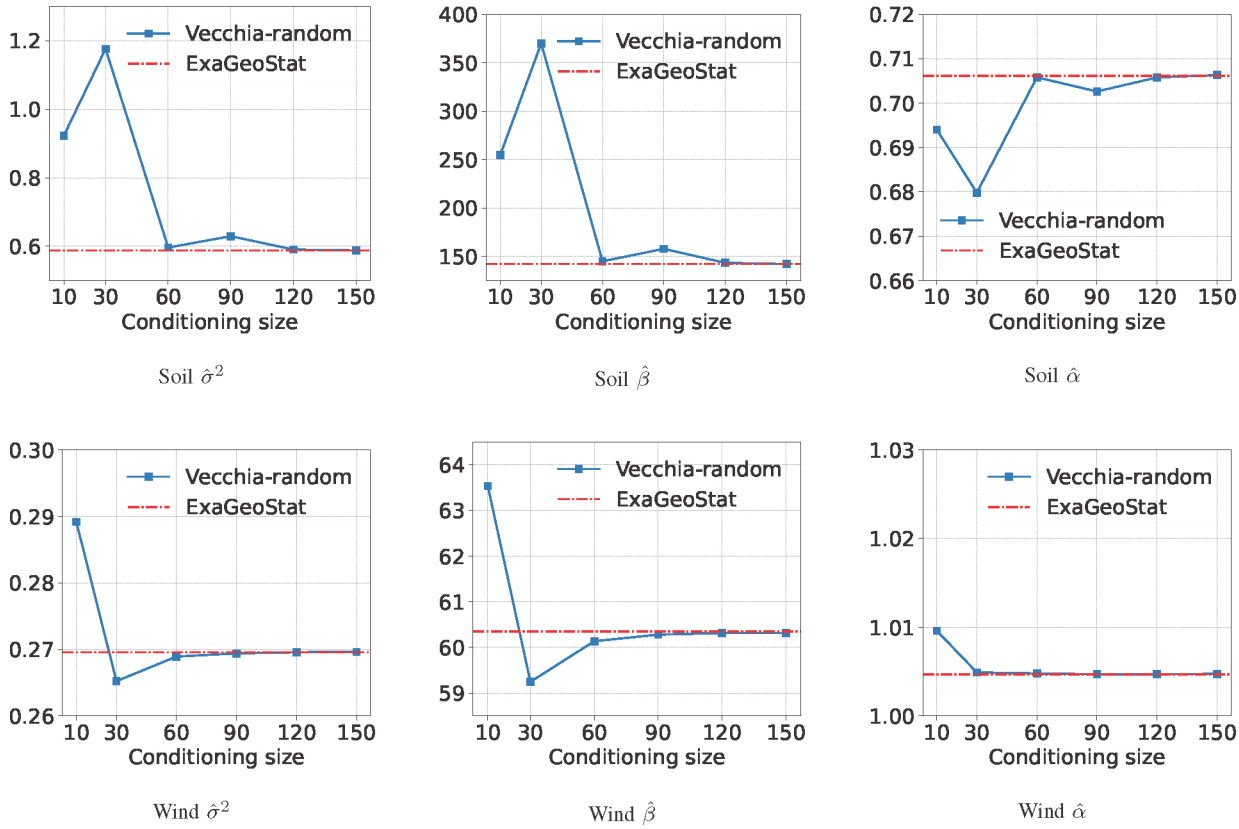


Fig. 8: The estimated parameter vectors using Vecchia approximation with different conditioning sizes compared to *ExaGeoStat* (exact MLE). The first row is the parameter vector for soil moisture, and the second for wind speed.

size is the largest matrix dimension that can fully fit into the GPU memory for exact MLE. Moreover, considering the space complexity, the batched Vecchia approximation can handle larger problem sizes of up to 1 million in a single GPU.

VII. Conclusions

Gaussian processes (GPs) are a powerful and flexible tool used in statistical modeling and machine learning for various tasks, including modeling, regression, and classification. However, GPs encounter a significant computational burden when dealing with high-dimensional data, prohibiting its use with massive amounts of data. Thus, many methods have been proposed to approximate the covariance matrix associated with the GPs. Among these methods is the Vecchia approximation algorithm, which allows a large-scale approximation of GPs.

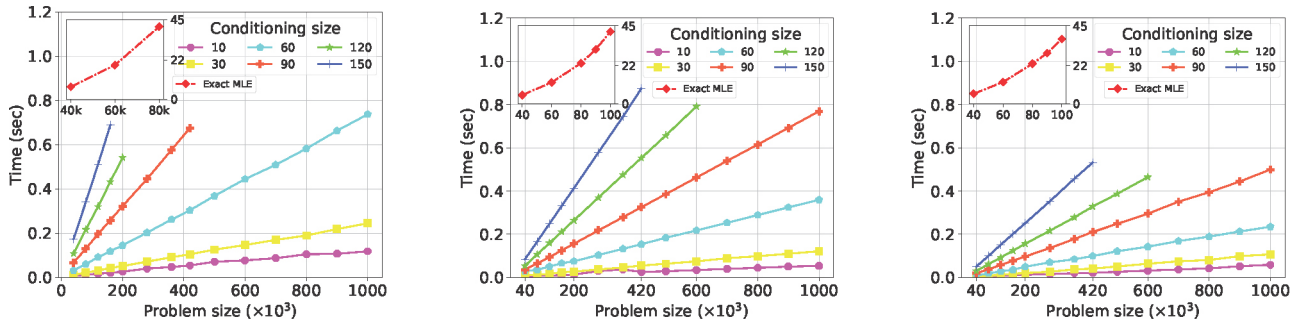
This work presents a parallel implementation of the Vecchia approximation technique that utilizes batched matrix computations on modern GPUs. Using batched linear algebra operations and the KBLAS library, the proposed implementation significantly reduces the time to solution compared to the state-of-the-art parallel implementation in the *ExaGeoStat* software. The speedup achieved on various GPU models (GV100, A100, H100) ranges from 700X to 1380X compared to the exact solution. The implementation can also manage larger problem sizes, accommodating up to 1 million geospatial locations with 80GB A100 and H100 GPUs while maintaining accuracy. The study also assesses the accuracy performance of the Vecchia approximation algorithm on real geospatial datasets, specifically soil moisture data in the Mississippi Basin area and wind speed data in the Middle East. The code can be found at: https://github.com/kaust-es/ParallelVecchiaGP.

Acknowledgment

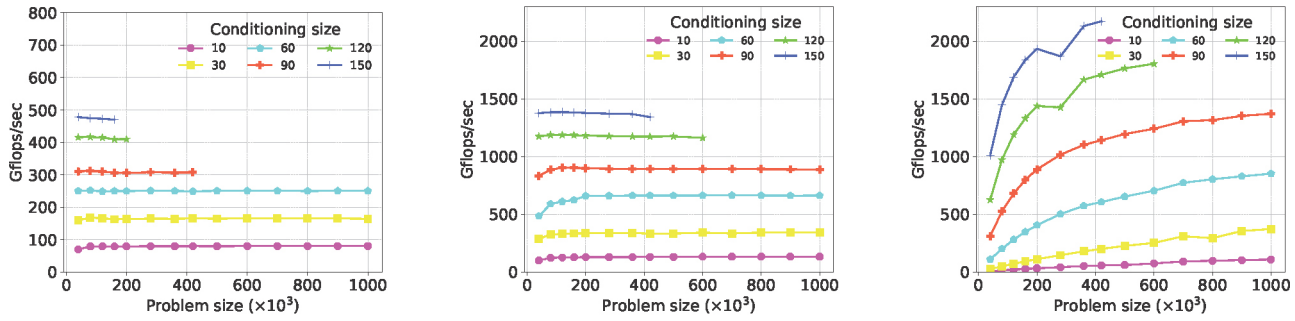
This research received support from the King Abdullah University of Science and Technology (KAUST) in Saudi Arabia. Our gratitude extends to the team at the KAUST Supercomputing Laboratory (KSL) and the Extreme Computing Research Center (ECRC) for providing the computational resources that were essential for the experiments conducted in this study. We extend our gratitude to Jie Ren (ECRC/KAUST) and Mohsin Shaikh (KSL/KAUST) for their assistance throughout the project.

References

[1] 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.



NVIDIA GV100 GPU with 32GB. NVIDIA A100 GPU with 80GB. NVIDIA H100 GPU with 80GB.



NVIDIA GV100 GPU with 32GB. NVIDIA A100 GPU with 80GB. NVIDIA H100 GPU with 80GB.

Fig. 9: Evaluation of computational performance on various NVIDIA GPUs, i.e., GV100, A100, and H100: the first row indicates the execution time required for a single likelihood estimation, where the small subfigures represent the performance of *ExaGeoStat*. The second row shows the Gflops/sec achieved by our implementation for a single likelihood estimation.

[2] 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.

[3] M. Bevilacqua, A. Fassò, C. Gaetan, E. Porcu, and D. Velandia, "Covariance tapering for multivariate Gaussian random fields estimation," *Statistical Methods & Applications*, vol. 25, pp. 21–37, 2016.

[4] D. Nychka, S. Bandyopadhyay, D. Hammerling, F. Lindgren, and S. Sain, "A multiresolution Gaussian process model for the analysis of large spatial datasets," *Journal of Computational and Graphical Statistics*, vol. 24, no. 2, pp. 579–599, 2015.

[5] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "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.

[6] Abdulah, Sameh and Ltaief, Hatem and Sun, Ying and Genton, Marc G and Keyes, David E, "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.

[7] 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.

[8] 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.

[9] M. Katzfuss and N. Cressie, "Spatio-temporal smoothing and em estimation for massive remote-sensing data sets," *Journal of Time Series Analysis*, vol. 32, no. 4, pp. 430–446, 2011.

[10] H. Huang and Y. Sun, "Hierarchical low rank approximation of likelihoods for large spatial datasets," *Journal of Computational and Graphical Statistics*, vol. 27, no. 1, pp. 110–118, 2018.

[11] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Parallel approximation of the maximum likelihood estimation for the prediction of large-scale geostatistics simulations," in *2018 IEEE international conference on cluster computing (CLUSTER)*. IEEE, 2018, pp. 98–108.

[12] S. Mondal, S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Parallel approximations of the tukey g-and-h likelihoods and predictions for non-Gaussian geostatistics," in *2022 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2022, pp. 379–389.

[13] A. V. Vecchia, "Estimation and model identification for continuous spatial processes," *Journal of the Royal Statistical Society Series B: Statistical Methodology*, vol. 50, no. 2, pp. 297–312, 1988.

[14] M. Katzfuss and J. Guinness, "A general framework for Vecchia approximations of Gaussian processes," 2021.

[15] M. Katzfuss, J. Guinness, and E. Lawrence, "Scaled Vecchia approximation for fast computer-model emulation," *SIAM/ASA Journal on Uncertainty Quantification*, vol. 10, no. 2, pp. 537–554, 2022.

[16] J. Zhang and M. Katzfuss, "Multi-scale Vecchia approximations of Gaussian processes," *Journal of Agricultural, Biological and Environmental Statistics*, vol. 27, no. 3, pp. 440–460, 2022.

[17] F. Jimenez and M. Katzfuss, "Scalable Bayesian optimization using vecchia approximations of Gaussian processes," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2023, pp. 1492–1512.

[18] "The Top 500 List," https://top500.org/, [Online; accessed 28-November-2023].

[19] A. Haidar, T. Dong, P. Luszczek, S. Tomov, and J. Dongarra, "Batched matrix computations on hardware accelerators based on GPUs," *The International Journal of High Performance Computing Applications*, vol. 29, no. 2, pp. 193–208, 2015.

[20] A. Abdelfattah, S. Tomov, and J. Dongarra, "Fast batched matrix mul-

tiplication for small sizes using half-precision arithmetic on GPUs," in *2019 IEEE international parallel and distributed processing symposium (IPDPS)*. IEEE, 2019, pp. 111–122.

[21] J. Guinness, "Permutation and grouping methods for sharpening Gaussian process approximations," *Technometrics*, vol. 60, no. 4, pp. 415–429, 2018.

[22] Guinness, Joseph, "Gaussian process learning via Fisher scoring of Vecchia's approximation," *Statistics and Computing*, vol. 31, no. 3, p. 25, 2021.

[23] 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.

[24] S. Abdulah, F. Alamri, P. Nag, Y. Sun, H. Ltaief, D. E. Keyes, and M. G. Genton, "The second competition on spatial statistics for large datasets," *arXiv preprint arXiv:2211.03119*, 2022.

[25] Y. Hong, Y. Song, S. Abdulah, Y. Sun, H. Ltaief, D. E. Keyes, and M. G. Genton, "The third competition on spatial statistics for large datasets," *Journal of Agricultural, Biological and Environmental Statistics*, pp. 1–18, 2023.

[26] R. Huser, M. L. Stein, and P. Zhong, "Vecchia likelihood approximation for accurate and fast inference in intractable spatial extremes models," *arXiv preprint arXiv:2203.05626*, 2022.

[27] Q. Vu, A. Zammit-Mangion, and S. J. Chuter, "Constructing large non-stationary spatio-temporal covariance models via compositional warpings," *Spatial Statistics*, vol. 54, p. 100742, 2023.

[28] J. Zhang, S. You, and L. Gruenwald, "Large-scale spatial data processing on GPUs and GPU-accelerated clusters," *Sigspatial Special*, vol. 6, no. 3, pp. 27–34, 2015.

[29] X. Li, T. Huang, D.-T. Lu, and C. Niu, "Accelerating experimental high-order spatial statistics calculations using GPUs," *Computers & Geosciences*, vol. 70, pp. 128–137, 2014.

[30] J. Zhang, S. You, and L. Gruenwald, "Efficient parallel zonal statistics on large-scale global biodiversity data on GPUs," in *Proceedings of the 4th International ACM SIGSPATIAL Workshop on Analytics for Big Geospatial Data*, 2015, pp. 35–44.

[31] G. Zhang, A.-X. Zhu, and Q. Huang, "A GPU-accelerated adaptive kernel density estimation approach for efficient point pattern analysis on spatial big data," *International Journal of Geographical Information Science*, vol. 31, no. 10, pp. 2068–2097, 2017.

[32] S. K. Prasad, M. McDermott, S. Puri, D. Shah, D. Aghajarian, S. Shekhar, and X. Zhou, "A vision for GPU-accelerated parallel computation on geo-spatial datasets," *SIGSPATIAL Special*, vol. 6, no. 3, pp. 19–26, 2015.

[33] K. Wang, S. Abdulah, Y. Sun, and M. G. Genton, "Which parameterization of the Matérn covariance function?" *Spatial Statistics*, vol. 58, p. 100787, 2023.

[34] J. Duchi, "Derivations for linear algebra and optimization," *Berkeley, California*, vol. 3, no. 1, pp. 2325–5870, 2007.

[35] E. Anderson, Z. Bai, C. Bischof, L. S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney *et al.*, *LAPACK users' guide*. SIAM, 1999.

[36] M. Gates, J. Kurzak, A. Charara, A. YarKhan, and J. Dongarra, "SLATE: Design of a modern distributed and accelerated linear algebra library," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2019, pp. 1–18.

[37] J. Dongarra, M. Gates, A. Haidar, J. Kurzak, P. Luszczek, P. Wu, I. Yamazaki, A. YarKhan, M. Abalenkovs, N. Bagherpour *et al.*, "PLASMA: Parallel linear algebra software for multicore using openmp," *ACM Transactions on Mathematical Software (TOMS)*, vol. 45, no. 2, pp. 1–35, 2019.

[38] S. Abdulah, K. Akbudak, W. Boukaram, A. Charara, D. Keyes, H. Ltaief, A. Mikhalev, D. Sukkari, and G. Turkiyyah, "Hierarchical computations on manycore architectures (hicma)," *See http://github. com/ecrc/hicma*, 2019.

[39] S. Zampini, W. Boukaram, G. Turkiyyah, O. Knio, and D. Keyes, "H2opus: a distributed-memory multi-GPU software package for non-local operators," *Advances in Computational Mathematics*, vol. 48, no. 3, p. 31, 2022.

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

[41] A. Abdelfattah, D. Keyes, and H. Ltaief, "KBLAS: An optimized library for dense matrix-vector multiplication on GPU accelerators," *ACM Transactions on Mathematical Software (TOMS)*, vol. 42, no. 3, pp. 1–31, 2016.

[42] F. G. Van Zee and R. A. Van De Geijn, "BLIS: A framework for rapidly instantiating BLAS functionality," *ACM Transactions on Mathematical Software (TOMS)*, vol. 41, no. 3, pp. 1–33, 2015.

[43] T. Dong, A. Haidar, P. Luszczek, S. Tomov, A. Abdelfattah, and J. Dongarra, "MAGMA batched: A batched BLAS approach for small matrix factorizations and applications on GPUs," Technical Report. Technical report, Tech. Rep., 2016.

[44] J. Dongarra, S. Hammarling, N. J. Higham, S. D. Relton, P. Valero-Lara, and M. Zounon, "The design and performance of batched BLAS on modern high-performance computing systems," *Procedia Computer Science*, vol. 108, pp. 495–504, 2017.

[45] K. Akbudak, H. Ltaief, A. Mikhalev, and D. Keyes, "Tile low rank cholesky factorization for climate/weather modeling applications on manycore architectures," in *International Conference on High Performance Computing*. Springer, 2017, pp. 22–40.

[46] Z. Geng, S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "GPU-accelerated dense covariance matrix generation for spatial statistics applications," 2023.

[47] N. W. Chaney, P. Metcalfe, and E. F. Wood, "HydroBlocks: A field-scale resolving land surface model for application over continental extents," *Hydrological Processes*, vol. 30, no. 20, pp. 3543–3559, 2016.

[48] J. Powers, X.-Y. Huang, B. Klemp, C. Skamarock, J. Dudhia, and O. Gill, "A description of the advanced research WRF version 2," *NCAR tech*, vol. 15, 2008.