

Scaled Block Vecchia Approximation for High-Dimensional Gaussian Process Emulation on GPUs

Qilong Pan, Sameh Abdulah¹, Senior Member, IEEE, Mustafa Abduljabbar, Hatem Ltaief², Andreas Herten³, Mathis Bode³, Matthew Pratola, Arindam Fadikar, Marc G. Genton⁴, David E. Keyes⁵, and Ying Sun⁶

Abstract—Emulating computationally intensive scientific simulations is crucial for enabling uncertainty quantification, optimization, and informed decision-making at scale. Gaussian Processes (GPs) offer a flexible and data-efficient foundation for statistical emulation, but their poor scalability limits applicability to large datasets. We introduce the Scaled Block Vecchia (SBV) algorithm for distributed GPU-based systems. SBV integrates the Scaled Vecchia approach for anisotropic input scaling with the Block Vecchia (BV) method to reduce computational and memory complexity while leveraging GPU acceleration techniques for efficient linear algebra operations. To the best of our knowledge, this is the first distributed implementation of any Vecchia-based GP variant. Our implementation employs MPI for inter-node parallelism and the MAGMA library for GPU-accelerated batched matrix computations. We demonstrate the scalability and efficiency of the proposed algorithm through experiments on both synthetic and real-world workloads, including a 50M-point simulation from a respiratory disease model. SBV achieves near-linear scalability on up to 512 A100 and GH200 GPUs, handles 2.56 B points, and reduces energy use relative to exact GP solvers, establishing SBV as a scalable and energy-efficient framework for emulating large-scale scientific models on GPU-based distributed systems.

Index Terms—Batched linear algebra, energy-efficient HPC, Gaussian processes, GPU computing, high-dimensional data, scalable emulation, vecchia approximation.

I. INTRODUCTION

MANY modern scientific and engineering advances rely on the use of computer models that use mathematical equations to represent physical systems and are implemented as high-fidelity simulations. These models allow researchers to explore phenomena that are impractical to observe experimentally, such as climate dynamics [1], [2], materials [3], [4], and cosmological structure formation [5], [6], [7]. Computer models can generate predictions that inform policy decisions, drive technological innovation, and facilitate fundamental discoveries by solving systems of differential equations or leveraging particle-based methods [1], [3], [5]. However, when exploring models with many input parameters, learning the relationship between inputs and outputs may require thousands of evaluations [8]. Such high computational costs often constrain their usage, motivating the adoption of statistical emulators to approximate the simulator's behavior at a fraction of the cost [9], [10].

A common approach to building computer model emulators is to fit a Gaussian Process (GP) to data generated by running the simulator at a predefined set of input values (design points), thereby learning a mapping from the model input space to the model output space [11]. GP-based emulators enhance interpretability and facilitate efficient analysis, particularly when simulating scenarios across a broad parameter space. For example, variable importance [12] and sequential optimization [13] may be feasible only with a cheap-to-evaluate GP emulator of the underlying expensive-to-evaluate simulator. While methods such as RBF-based models [14], [15] can offer flexible function approximation for such endeavors, they are fundamentally deterministic heuristics that lack marginal likelihood formulations, statistically grounded parameter estimation, and coherent uncertainty quantification. As a result, they are not well aligned with the probabilistic and inference-driven objectives of this work. In contrast, GP modeling provides a fully probabilistic framework with principled uncertainty quantification and likelihood-based inference, which are essential when studying scalable approaches to inference and are made possible by covariance approximations such as Vecchia-based methods [16], [17].

However, GPs face significant scalability challenges due to their $\mathcal{O}(n^3)$ computational cost and $\mathcal{O}(n^2)$ memory requirements, where n is the number of data points. These costs arise from operations on a dense $n \times n$ covariance matrix that

Received 18 August 2025; revised 31 August 2026; accepted 5 September 2026. Date of publication 8 September 2026; date of current version 6 October 2026. This work was supported by the King Abdullah University of Science and Technology (KAUST), Thuwal 23955, Saudi Arabia. Recommended for acceptance by Z. Xu. (Corresponding author: Ying Sun.)

Qilong Pan, Marc G. Genton, and Ying Sun are with the Statistics Program, King Abdullah University of Science and Technology (KAUST), Thuwal 23955, Saudi Arabia (e-mail: qilong.pan@kaust.edu.sa; marc.genton@kaust.edu.sa; ying.sun@kaust.edu.sa).

Sameh Abdulah, Mustafa Abduljabbar, Hatem Ltaief, and David E. Keyes are with the Applied Mathematics and Computational Science (AMCS) Program, King Abdullah University of Science and Technology (KAUST), Thuwal 23955, Saudi Arabia (e-mail: sameh.abdulah@kaust.edu.sa; mustafa.abduljabbar@kaust.edu.sa; hatem.ltaief@kaust.edu.sa; david.keyes@kaust.edu.sa).

Andreas Herten and Mathis Bode are with Jülich Supercomputing Centre (JSC), 52428 Forschungszentrum Jülich, Germany (e-mail: a.herten@fz-juelich.de; m.bode@fz-juelich.de).

Matthew Pratola is with the Department of Statistics, Indiana University, Bloomington, IN 47405 USA (e-mail: mpratola@iu.edu).

Arindam Fadikar is with Argonne National Laboratory, Lemont, IL 60439 USA (e-mail: afadikar@anl.gov).

Digital Object Identifier 10.1109/TPDS.2026.3732178

captures the correlations between data inputs [18]. For instance, modeling 200 K locations demands about 300 GB of memory and an estimated 2.6 PFLOP of computation, making standard GPs infeasible for large-scale applications without approximation methods. Thus, numerous studies have addressed the computational and memory challenges of large-scale GPs. Some efforts have pursued exact GP computation on High-Performance Computing (HPC) systems [18], [19], [20]. For instance, the largest reported problem size was processed on ORNL's Frontier system, handling 27.24M data points in mixed-precision across approximately 9K nodes (~ 36 K AMD GPUs) of the system [10]. However, most of the literature has focused on approximation to alleviate the computational burden of GP modeling, including low-rank [21], [22], [23], tile low-rank [24], [25], [26], covariance tapering [21], and Vecchia approximations [17], [27].

Among these approximation methods, Classic Vecchia (CV) approximation demonstrates superior advantages. In [28], Katzfuss et al. introduced the Vecchia approximation for GP emulation in the context of computer experiments and proposed the Scaled Vecchia (SV) method to enhance approximation accuracy in high-dimensional settings. However, their study was limited to small-scale problems and relied entirely on CPU-based computations. In contrast, efforts have been made to accelerate the classic Vecchia algorithm on GPUs, as in [27], which subsequently introduced the Block Vecchia (BV) approach [29] to further improve computational speed while preserving approximation accuracy. Nevertheless, the use of CV approximation for high-dimensional GP emulation on GPUs remains unexplored. This gap presents an opportunity to develop a scalable GPU-based method that can efficiently address the computational and memory challenges posed by high-dimensional inputs.

In this work, we propose a novel algorithm, the Scaled Block Vecchia (SBV) approximation, for distributed GPU-based systems. SBV integrates the Block Vecchia (BV) algorithm from [29] with the Scaled Vecchia approach introduced in [28], while leveraging GPU acceleration techniques from [27]. To the best of our knowledge, this work represents the first distributed implementation of any variant of the Vecchia-based GP algorithm. Leveraging the Message Passing Interface (MPI), we parallelize computations across multiple GPU-enabled nodes, achieving scalable performance well beyond that demonstrated in prior studies.

To validate the accuracy and robustness of our algorithm, we perform GP simulations across a range of parameter settings and benchmark its performance on the satellite drag dataset, a standard high-dimensional testbed for evaluating GP-based models [30]. As an application in computer model emulation, we apply our algorithm to *MetaRVM*, a compartmental simulation model designed for generic respiratory virus diseases [31]. The accuracy experiments include end-to-end fitting and prediction on 50M *MetaRVM* samples, whereas separate distributed scaling studies use workloads capped at 500 likelihood-objective evaluations across up to 512 A100/GH200 GPUs. The reported capped GH200 weak-scaling run reaches 2.56 B samples across 512 GPUs, demonstrating the implementation's ability to accommodate and execute SBV likelihood evaluations at this scale and

indicating its potential for extremely large-scale applications. Additionally, our approach achieves a noticeable reduction in power consumption compared to the state-of-the-art exact GP framework on GPUs, i.e., *ExaGeoStat* [18].

The paper is organized as follows: Section II summarizes the main contributions of the study. Section III provides a brief overview of related work. Section IV presents the necessary background. Section V details the proposed algorithm. Section VI describes the simulation study and reports the results of the computer model application. Section VII focuses on performance evaluation and analysis. Finally, conclusions are presented in Section VIII.

II. CONTRIBUTIONS

The key contributions of this work are as follows:

- We introduce Scaled Block Vecchia (SBV), a Gaussian process approximation that combines anisotropic input scaling with block conditioning to enable accurate and scalable emulation for high-dimensional, large datasets.
- We develop Random Anchor Clustering (RAC) and a filtered nearest-neighbor search pipeline that reduce the cost of block construction and neighbor selection, making SBV practical for very large high-dimensional problems.
- We develop a distributed GPU implementation of SBV using MPI for inter-node parallelism and batched MAGMA kernels [32], [33] for high-throughput linear algebra.
- We evaluate SBV on synthetic, satellite-drag, and 50M-point *MetaRVM* workloads, demonstrating strong predictive accuracy and computational performance across representative high-dimensional emulation problems.
- We demonstrate weak and strong scaling on A100- and GH200-based systems up to 512 GPUs and problem sizes of 2.56 B points, together with a power and energy analysis relative to exact GP computation.

Section V explicitly maps the main integration challenges to their corresponding algorithmic components.

III. RELATED WORK

GP emulators: GPs are widely used for emulating high-fidelity simulators in scientific computing, with applications in cosmology [6], climate modeling [10], and mechanism science [30]. Notable examples include emulation of N-body cosmological simulations [7], particle-material interactions [4], and epidemiological models of disease spread [34]. While recent advances in scalable GPs have improved performance, few approaches offer distributed, multi-GPU implementations. The proposed SBV algorithm directly addresses this gap by enabling efficient, scalable GP emulation on distributed, GPU-accelerated HPC systems.

CV approximation: CV approximation [16], [17] reduces the computational complexity of GPs by factorizing the joint distribution into a product of conditional distributions. Recent advancements have enhanced the effectiveness of the CV approximation by improving data-point ordering and neighbor selection [35]. The BV approximation [29] builds on this by conditioning on blocks of observations rather than individual

points, enabling batched linear algebra operations on GPUs and improving throughput and memory efficiency in high-performance environments [27]. More recently, the SV method was introduced to address challenges associated with high-dimensional inputs by applying anisotropic input scaling [28], which enables the model to capture directionally varying correlations. Our work extends these developments by integrating block-based conditioning and anisotropic scaling into a unified framework, i.e., SBV, and presenting a distributed implementation optimized for multi-GPU systems.

Vecchia-Based Applications and Extensions: Recent research has increasingly adopted the Vecchia approximation as a scalable computational primitive within broader probabilistic modeling frameworks. For example, Vecchia approximations have been incorporated into deep Gaussian processes to enable fully Bayesian posterior inference with near-linear scaling for large computer experiments [36]. In the context of latent Gaussian process models with non-Gaussian likelihoods, iterative methods have been developed to accelerate Vecchia-Laplace approximations using conjugate gradients and stochastic trace estimation [37]. It has also been extended to scalable Gaussian process regression with variable selection via penalized-likelihood formulations and mini-batch subsampling [38], and adapted for large-scale Bayesian optimization by integrating approximate nearest-neighbor search and variance-recalibration strategies [39]. Beyond regression settings, Vecchia likelihood approximations have been employed for computationally tractable inference in high-dimensional spatial max-stable models [40], and have recently enabled linear-cost estimation of multivariate normal probabilities via sparse inverse Cholesky structures [41]. Previous studies have shown that the Vecchia approximation is a versatile and scalable tool for probabilistic modeling in various statistical fields. In contrast, our work pursues a different goal. We focus on likelihood-based Gaussian process emulation for high-dimensional inputs, combining scaled and block conditioning strategies, distributed-memory parallelization using MPI, GPU-accelerated batched dense linear algebra, and energy-efficient analysis on modern GPU supercomputers with billion-scale datasets. To our knowledge, this is the first distributed GPU implementation of a Vecchia-based Gaussian process method that scales to billions of data points, advancing Vecchia approximations from methodological scalability to extreme-scale HPC through accelerators.

IV. BACKGROUND

This section provides background on GPs, the BV approximation, Kullback–Leibler (KL) divergence, and the scaled kernel function, which together form the foundation for our proposed extension for high-dimensional data. Table I summarizes the abbreviations used throughout the paper to enhance clarity and readability.

A. Gaussian Processes (GPs)

Throughout, vectors, matrices, and point/block sets are denoted in bold, whereas scalars and indices are unbolded. GPs provide a flexible framework for modeling and predicting

TABLE I
LIST OF ABBREVIATIONS AND THEIR EXPLANATIONS

Abbreviation	Explanation
GPs	Gaussian Processes
CV/SV/BV/SBV	Classic/Scaled/Block/Scaled Block Vecchia
n/n^*	Problem size in estimation/prediction
bs_{est}/bc_{est} (bs/bc)	Block (batched) size/count for estimation
bs_{pred}/bc_{pred} (bs^*/bc^*)	Block (batched) size/count for prediction
m_{est}/m_{pred}	# nearest neighbors for estimation/prediction
$P; p, q$	# workers; worker indices
$K = bc; k_p$	Total block count; local blocks/anchors on worker p
$\alpha; \lambda$	Candidate expansion factor; distance threshold

functions across low- and high-dimensional input spaces. Let $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_n]^\top$, where each $\mathbf{x}_i \in \mathbb{R}^d$, denote the n input points with corresponding observations $\mathbf{y} = [y_1, y_2, \dots, y_n]^\top$. A GP with zero mean and kernel $K_\theta(\mathbf{x}_i, \mathbf{x}_j)$ is formulated as $\mathbf{y} \sim \mathcal{N}(\mathbf{0}, \Sigma_\theta)$ and its log-likelihood can be represented as shown in [42]:

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

where $|\cdot|$ means determinant, $\Sigma_\theta \in \mathbb{R}^{n \times n}$ has entries $[\Sigma_\theta]_{ij} = K_\theta(\mathbf{x}_i, \mathbf{x}_j)$ parameterized by θ . For a set of n^* new input points $\mathbf{X}^* = [\mathbf{x}_1^*, \mathbf{x}_2^*, \dots, \mathbf{x}_{n^*}^*]^\top$, we are interested in predicting their corresponding observations $\mathbf{y}^* = [y_1^*, y_2^*, \dots, y_{n^*}^*]^\top$. The joint distribution of $\mathbf{y}$ and $\mathbf{y}^*$ is assumed to be:

$$\begin{bmatrix} \mathbf{y} \\ \mathbf{y}^* \end{bmatrix} \sim \mathcal{N} \left(\mathbf{0}, \begin{bmatrix} \Sigma_\theta & \Sigma_{\theta,*} \\ \Sigma_{\theta,*}^\top & \Sigma_\theta^* \end{bmatrix} \right),$$

where $\Sigma_{\theta,*} \in \mathbb{R}^{n \times n^*}$ contains the covariance matrix between $\mathbf{X}$ and $\mathbf{X}^*$, and $\Sigma_\theta^* \in \mathbb{R}^{n^* \times n^*}$ contains the covariances among $\mathbf{X}^*$. The conditional distribution of $\mathbf{y}^*$ given $\mathbf{y}$ is:

$$\mathbf{y}^* | \mathbf{y} \sim \mathcal{N}(\boldsymbol{\mu}^*, \Sigma^*),$$

where the conditional mean $\boldsymbol{\mu}^*$ and covariance Σ^* are derived as $\boldsymbol{\mu}^* = \Sigma_{\theta,*}^\top \Sigma_\theta^{-1} \mathbf{y}$, $\Sigma^* = \Sigma_\theta^* - \Sigma_{\theta,*}^\top \Sigma_\theta^{-1} \Sigma_{\theta,*}$. The GPs have computational complexity $\mathcal{O}(n^3)$ and memory complexity $\mathcal{O}(n^2)$, making them infeasible for large-scale problems [42], [16].

B. Block Vecchia Method

The BV approximation [27] reduces the computational and memory cost of GPs by partitioning/clustering the dataset into bc disjoint blocks, $\{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_{bc}\}$. After applying a permutation ζ to order the blocks, the exact likelihood of $\mathbf{y}$ becomes:

$$p_\theta(\mathbf{y}) = \prod_{i=1}^{bc} p_\theta \left(\mathbf{y}_{\mathbf{B}_i^\zeta} | \mathbf{y}_{\mathbf{B}_1^\zeta}, \mathbf{y}_{\mathbf{B}_2^\zeta}, \dots, \mathbf{y}_{\mathbf{B}_{i-1}^\zeta} \right).$$

The BV method approximates each conditional distribution by replacing each vector with a subset of neighbors $NN(\mathbf{B}_i^\zeta)$ with:

$$p_\theta(\mathbf{y}) \approx \prod_{i=1}^{bc} p_\theta \left(\mathbf{y}_{\mathbf{B}_i^\zeta} | \mathbf{y}_{NN(\mathbf{B}_i^\zeta)} \right), \quad (2)$$

where the number of $NN(\mathbf{B}_i^\zeta)$ is m and $m \ll n$. The prediction for $\mathbf{y}^*$ under the BV framework approximates the conditional

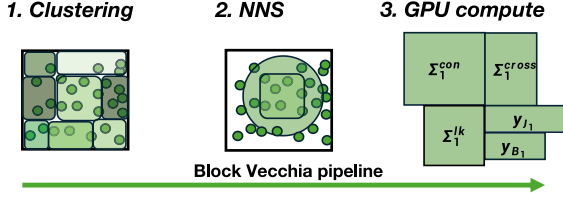


Fig. 1. The BV algorithm: (1) disjoint clustering, (2) Nearest Neighbor Searching (NNS), and (3) batched GPU log-likelihood computation.

distribution $p(\mathbf{y}^*|\mathbf{y})$ by the product of block-level conditional distributions. Given the new blocks $\{\mathbf{B}_1^*, \dots, \mathbf{B}_{bc^*}^*\}$ on $\mathbf{X}^*$, the predictive distribution is:

$$p_{\theta}(\mathbf{y}^*|\mathbf{y}) \approx \prod_{j=1}^{bc^*} p_{\theta}(\mathbf{y}_{\mathbf{B}_j^*} | \mathbf{y}_{NN(\mathbf{B}_j^*)}), \quad (3)$$

where $NN(\mathbf{B}_j^*)$ contains the neighbors selected from the $\mathbf{y}$. The BV method reduces the computational complexity of the exact GPs from $\mathcal{O}(n^3)$ to $\mathcal{O}(bc \cdot m^3)$ and memory complexity from $\mathcal{O}(n^2)$ to $\mathcal{O}(bc \cdot m^2)$. Fig. 1 shows the computations involved in the BV algorithm [27], which includes three components, clustering $\{\mathbf{B}_1, \mathbf{B}_2, \dots, \mathbf{B}_{bc}\}$, Nearest Neighbor Searching (NNS) $NN(\mathbf{B}_i^{\zeta})$, and batched GPU computing.

C. Kullback-Leibler (KL) Divergence

The KL divergence measures how one probability distribution diverges from a second probability distribution [43]. For two Gaussian distributions $\mathcal{N}_0$ and $\mathcal{N}_1$ with zero mean and covariance matrices Σ_0 and Σ_1 , the KL divergence is [44]:

$$D_{KL}(\mathcal{N}_0||\mathcal{N}_1) = \frac{1}{2} \left\{ \text{tr}(\Sigma_1^{-1}\Sigma_0) - n + \log \left| \frac{\Sigma_1}{\Sigma_0} \right| \right\}.$$

For Vecchia-based GPs, previous work [27], [29] shows that the KL divergence simplifies to the difference between exact and approximate log-likelihoods evaluated at $\mathbf{y} = \mathbf{0}$ as:

$$D_{KL} = \ell_0(\theta; \mathbf{0}) - \ell_a(\theta; \mathbf{0}), \quad (4)$$

where $\ell_0(\theta; \mathbf{0})$ is the exact log-likelihood and $\ell_a(\theta; \mathbf{0})$ is Vecchia-based approximate log-likelihood.

This work uses the KL divergence to quantify the discrepancy between the proposed SBV algorithm and posterior distributions from other GP methods.

D. Scaled Kernel Function

In high-dimensional spaces, covariance functions often require *anisotropic scaling*, where each input dimension is scaled according to its relevance to the output [28]. This allows the model to reflect varying sensitivities across dimensions, thereby improving both accuracy and efficiency. The scaled covariance function is defined as

$$K_{\theta}(\mathbf{x}_k, \mathbf{x}_{k'}) = f \left(\left(\sum_{i=1}^d \frac{(x_{ki} - x_{k'i})^2}{\beta_i^2} \right)^{\frac{1}{2}} \right), \quad (5)$$

where x_{ki} and $x_{k'i}$ denote the i -th components of $\mathbf{x}_k$ and $\mathbf{x}_{k'}$, and $\beta = (\beta_1, \dots, \beta_d)^{\top}$ contains the dimension-specific range (scaling) parameters. $f(\cdot)$ is a kernel function, such as the Matérn kernel:

$$f(r) = \sigma^2 \frac{2^{1-\nu}}{\Gamma(\nu)} r^{\nu} \mathcal{K}_{\nu}(r) + \sigma_0^2. \quad (6)$$

Here, r is the scaled distance and $\theta = (\sigma^2, \beta, \nu, \sigma_0^2)^{\top}$ denotes the kernel parameters: σ^2 (process variance), σ_0^2 (nugget), and $\nu > 0$ (smoothness). $\Gamma(\cdot)$ is the gamma function, and $\mathcal{K}_{\nu}(\cdot)$ is the modified Bessel function of the second kind of order ν . Additional valid covariance classes are detailed in [45]. We use the scaled form in Equation (5) to capture directional relevance, which is useful in emulation problems where only a subset of input dimensions significantly affects the output.

V. DISTRIBUTED SCALED BLOCK VECCHIA ALGORITHM

This section presents the proposed distributed Scaled Block Vecchia (SBV) algorithm. The algorithm comprises four key components: scaling and partitioning, Random Anchor Clustering (RAC), filtered subset selection for NNS, and batched GPU computation to optimize the log-likelihood. The preprocessing steps, i.e., scaling and partitioning, RAC, and filtered NNS, are performed once on the CPU to prepare the desired data structure. The resulting structured data is then transferred to the GPU, enabling efficient execution of hundreds of parameter optimization iterations without redundant data movement. The distributed SBV algorithm is detailed in Algorithm 1, and the pipeline is illustrated in Fig. 2, where Steps (2), (3), and (4) are the Block Vecchia (BV) algorithm pipeline.

Technical challenges and corresponding solutions: Four issues are nontrivial at distributed scale: (C1) scale-dependent geometry changes partitioning and conditioning candidates, addressed by scaled partitioning (Section V-A); (C2) block construction and unrestricted neighbor searches are expensive, addressed by RAC and filtered candidate selection (Sections V-B and V-C); (C3) conditioning data must be assembled across workers without centralizing the dataset, addressed by collective candidate exchange (Section V-C); and (C4) variable-size block conditionals must be evaluated repeatedly, addressed by resident GPU data, batched MAGMA kernels, and MPI_Allreduce (Section V-D).

Fig. 2 separates the workflow into one-time preprocessing and the repeated optimization loop. Scaling, point redistribution, RAC, and filtered m -NNS first construct the block and neighbor arrays that remain resident on the GPU. During each objective evaluation, the current θ is broadcast, rank-local batched kernels compute ℓ_p , and MPI_Allreduce forms the global log-likelihood used by the synchronized optimizer. Here, NLOpt is adopted to update the parameter set θ without using the gradient until the predefined accuracy is reached.

A. Data Partitioning and Scaling

This stage addresses the scale-dependent geometry in C1. Each worker p begins by loading its assigned data subset $\mathbf{X}_p$.

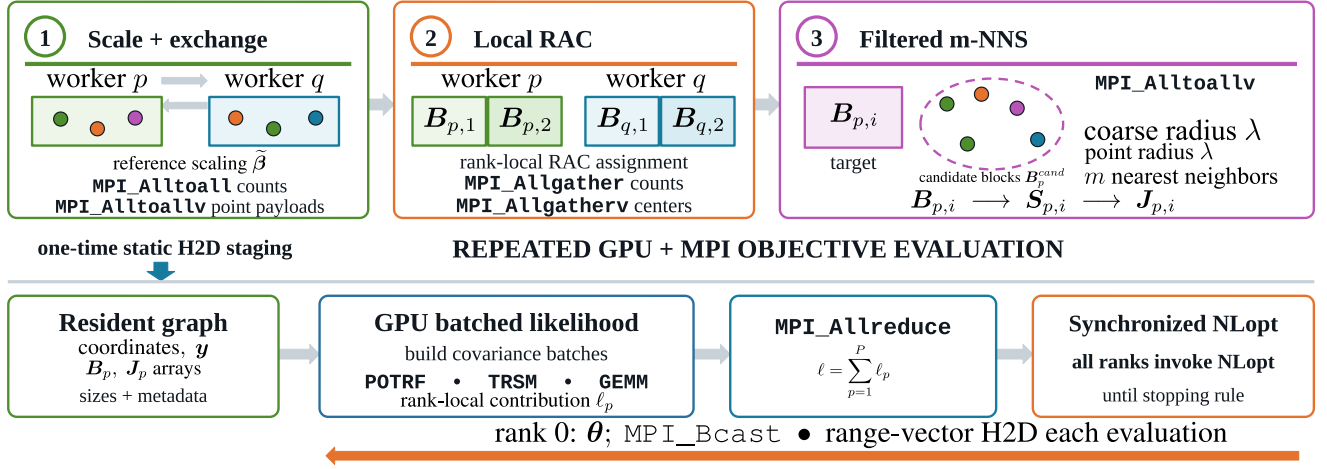


Fig. 2. Distributed SBV fitting workflow. One-time CPU preprocessing constructs and stages the block/neighbor data structure, where $\tilde{\beta}$ denotes the fixed reference scaling used for graph construction. Each likelihood evaluation broadcasts θ , executes the batched GPU kernels, and combines rank-local contributions using MPI_Allreduce. NLOpt [46], a nonlinear optimization library used for maximum-likelihood parameter estimation, runs synchronously across all MPI ranks using the globally reduced likelihood. Fig. 3 expands Step 3. Worker counts and geometry are schematic.

Algorithm 1: Distributed SBV approximation algorithm.

- 1: **Input:** Data $\{x_i, y_i\}_{i=1}^n$, dimension d , total block count K , nearest neighbors m_{est} , workers P , covariance function K_θ , fixed reference scaling $\tilde{\beta}$.
 - 2: **Output:** Approximate the log-likelihood ℓ .
 - 3: **Step 0: Data Loading** X_p^{org}
 - 4: Each worker loads local points X_p^{org} , $p = 1, 2, \dots, P$.
 - 5: **Step 1: Partitioning and Scaling** X_p^{org}
 - 6: $X_{1:P} = \mathcal{PS}(X_{1:P}^{\text{org}})$
 - 7: **Step 2: Random Anchor Clustering** B_p
 - 8: $\{B_{p,i}; p = 1, 2, \dots, P; i = 1, 2, \dots, k_p\} = \mathcal{C}(X_{1:P})$
 - 9: Randomly reorder these blocks $B_{p,i}$;
 - 10: **Step 3: NNS** J_p
 - 11: $\{J_{p,i}; p = 1, 2, \dots, P; i = 1, 2, \dots, k_p\} = \mathcal{V}(B_{1:P}, m_{\text{est}})$
 - 12: **Step 4: Batched Log-Likelihood Calculation** $\log \mathcal{L}_p$
 - 13: **for** each block $B_p, p = 1, 2, \dots, P$ in Parallel **do**
 - 14: $\ell_p = \ell_{\text{lh}}(B_p, J_p, y_{p,B}, y_{p,J})$, where $y_{p,B}, y_{p,J}$ are the observations associated with B_p and J_p respectively.
 - 15: **end for**
 - 16: **Step 5: Reduction of Log-Likelihood Across Workers**
 - 17: Use MPI_Allreduce to sum ℓ_p from each worker and obtain the approximated log-likelihood ℓ .
 - 18: **Return:** The approximated log-likelihood ℓ .
-

The points are partitioned along the most relevant dimension $d_{\max}$, as shown in Algorithm 2; MPI_Alltoall exchanges counts and MPI_Alltoallv redistributes point payloads. Input dimensions are then rescaled using the fixed reference vector $\tilde{\beta} = [\tilde{\beta}_1, \dots, \tilde{\beta}_d]$. This graph-building vector is distinct from the per-evaluation β in θ , which enters the covariance kernel without rebuilding the graph. Fig. 2 is schematic; its

Algorithm 2: Partitioning and Scaling $\mathcal{PS}(X_{1:P}^{\text{org}})$.

- 1: **Input:** Complete dataset $X_{1:P}^{\text{org}}$.
 - 2: **Output:** Scaled and redistributed $X_{1:P}$.
 - Relevant dimension
 - $d_{\max} = \arg \max_i \{1/\tilde{\beta}_i, i = 1, 2, \dots, d\}$.
 - 3: **for** each worker p in parallel **do**
 - 4: MPI_Alltoall exchanges destination counts.
 - 5: MPI_Alltoallv redistributes point payloads: send $x_{p,i}^{\text{org}}$ to q th node if $\text{int}(x_{p,i}^{\text{org}} \cdot P) = q$.
 - 6: **end for**
 - 7: **for** each worker p in parallel **do**
 - 8: Dimension scaling, $X_{p,j} := X_{p,j}^{\text{org}} / \tilde{\beta}_j$, $j = 1, 2, \dots, d$.
 - 9: **end for**
 - 10: **Return:** Redistributed and scaled $X_{1:P}$
-

two workers, block counts, and geometry illustrate the data flow rather than fixed settings.

B. Random Anchor Clustering (RAC)

RAC addresses the block-construction cost in C2. Next, small blocks B_p are created, defined as $B_p := \{B_{p,1}, \dots, B_{p,k_p}\}$, for each partition by RAC independently. The RAC method is detailed in Algorithm 3. Here, we replace K-means clustering in block Vecchia GPs [29] with RAC to reduce the computational cost of K-means on large datasets while maintaining comparable approximation accuracy. The RAC algorithm begins by randomly selecting k_p anchors (centers) for each worker $p = 1, 2, \dots, P$ in parallel. These anchors serve as the centers of blocks. Subsequently, each $x_{p,i}$ is assigned to the j th block based on its distance, as shown in line 5 of Algorithm 3, thus constructing clusters, $B_p := (B_{p,1}, B_{p,2}, \dots, B_{p,k_p})$ with $p = 1, 2, \dots, P$. The RAC method does not involve any

Algorithm 3: RAC algorithm $\mathcal{C}(\mathbf{X}_{1:P})$.

-
- 1: **Input:** $\mathbf{X}_{1:P}$ full dataset.
 - 2: **Output:** $\mathbf{B}_{1:P}$ set of clusters for each node.
 - 3: **for** each worker p in parallel **do**
 - 4: Randomly choose k_p local centers
 $\mathbf{C}_p = \{\mathbf{c}_{p,j} \mid j = 1, 2, \dots, k_p\}$ in $\mathbf{X}_p$, and initialize
 $\mathbf{B}_{p,j} = \{\mathbf{c}_{p,j}\}$.
 - 5: Assign $\mathbf{x}_{p,i}$ to cluster $\mathbf{B}_{p,j^*}$, where
 $j^* = \arg \min_{1 \leq j \leq k_p} \|\mathbf{x}_{p,i} - \mathbf{c}_{p,j}\|_2^2$.
 - 6: **end for**
 - 7: **Return:** Clusters $\mathbf{B}_{1:P}$
-

communication among workers, since the size of clusters is typically much smaller than the overall dataset. For example, a single cluster may contain only 100 locations, whereas the entire dataset comprises 10 million locations. The two blocks per worker in Fig. 2 are schematic rather than an algorithmic requirement.

C. Filtered m -Nearest Neighbor Searching (NNS)

This stage addresses candidate restriction (C2) and distributed conditioning-data exchange (C3). With block sets $\mathbf{B}_{1:P}$, the next goal is to find the exact m nearest neighbors for all block centers in high-dimensional space, where search areas vary for each query center in the Vecchia-based GPs; see Equation (2). Existing methods [47], [48], [49] for exact m -NNS with query-specific search areas face limitations, especially for large datasets.

To address these issues, we apply the filtered m -NNS to each block $\mathbf{B}_{p,i}$, as detailed in Algorithm 4. This approach selects a small filtered subset (a circle of radius λ) for brute-force m -NNS to obtain exact results while reducing computational cost. In Algorithms 1 and 4, $\mathcal{V}$ denotes the filtered m -NNS function, which begins by computing a Monte Carlo-based distance threshold. This threshold λ is calculated using:

$$\lambda = \left(\alpha \frac{m \zeta_d}{n} \right)^{1/d}, \quad \zeta_d = \begin{cases} \frac{\Gamma(\frac{d}{2}+1)}{\pi^{\frac{d}{2}}} & d \text{ is even} \\ \frac{2\pi^{\frac{d-1}{2}} \Gamma(\frac{d+1}{2})}{\Gamma(d+1)} & d \text{ is odd} \end{cases}. \quad (7)$$

Here, n is the total number of points, m is the number of nearest neighbors, d is the dimensionality, ζ_d is the geometric factor defined above, and α is an expansion factor to account for irregularities in point distribution (e.g., $\alpha = 100$ implies the candidate set is expected to be 100 times larger than m). Then we prepare the small filtered subset in two steps: coarse and fine preparation. In the coarser preparation, each worker redistributes candidate clusters to the corresponding worker if the distance between two centers is within the calculated threshold; `MPI_Alltoall` exchanges counts and `MPI_Alltoallv` redistributes block payloads. The complementary coarser candidate set is denoted as $\mathbf{B}_p^{cand}$ (lines 10–13 in Algorithm 4). In the finer partition, each worker independently selects finer candidates $\mathbf{S}_{p,i}$ for the i th cluster, i.e., $\mathbf{x}_k$ is added to $\mathbf{S}_{p,i}$ if the distance between the local center and $\mathbf{x}_k$ is within λ (lines 15–22 in Algorithm 4). Finally,

Algorithm 4. NNS algorithm based on filtered subset $\mathcal{V}(\mathbf{B}_p, m)$

-
- 1: **Input:** $\mathbf{B}_p$ cluster set, nearest neighbors m , expansion factor $\alpha = 100$ (default).
 - 2: **Output:** Nearest neighbor set $\mathbf{J}_{1:P}$
 - 3: **Step1: Distance Threshold and Centers**
 - 4: Calculate distance threshold, λ according to Equation (7).
 - 5: **for** each block $\mathbf{B}_{p,i}$ in $\mathbf{B}_p$ **do**
 - 6: Update center $\mathbf{c}_{p,i} = |\mathbf{B}_{p,i}|^{-1} \sum_{\mathbf{x} \in \mathbf{B}_{p,i}} \mathbf{x}$.
 - 7: **end for**
 - 8: `MPI_Allgather`, gathers all centers $\mathbf{C}_{1:P}$ and their values to each worker.
 - 9: **Step 2: Prepare Coarser Candidates**
 - 10: **for** each worker p in parallel **do**
 - 11: `MPI_Alltoall`, redistribute $\mathbf{B}_{p,i}$ to q th node if $\|\mathbf{c}_{p,i} - \mathbf{c}_{q,j}\|^2 \leq \lambda$ where j exists.
 - 12: Received clusters form the candidate set $\mathbf{B}_p^{cand}$ for worker p .
 - 13: **end for**
 - 14: **Step 3: Prepare Finer Candidates for NN**
 - 15: **for** local block center $\mathbf{c}_i$ in $\{\mathbf{c}_i\}_{i=1}^{k_p}$ **do**
 - 16: **for** candidate block center $\mathbf{c}_j$ in $\mathbf{B}_p^{cand}$ **do**
 - 17: **if** The order $\mathbf{c}_j$ not exceeds order $\mathbf{c}_i$ **then**
 - 18: **for** each point $\mathbf{x}_k$ in $\mathbf{B}_{p,j}^{cand}$ **do**
 - 19: Calculate the distance $d(\mathbf{c}_i, \mathbf{x}_k)$. If $d(\mathbf{c}_i, \mathbf{x}_k) < \lambda$, add $\mathbf{x}_k$ to the candidate set $\mathbf{S}_{p,i}$ for block $\mathbf{B}_{p,i}$.
 - 20: **end for**
 - 21: **end if**
 - 22: **end for**
 - 23: Searching m nearest neighbors for the i th block, i.e.,
 $\mathbf{J}_{p,i} = \text{NNS}(\mathbf{c}_i, \mathbf{S}_{p,i}, m)$.
 - 24: **end for**
 - 25: **Return:** Nearest neighbor set $\{\mathbf{J}_p, p = 1, 2, \dots, P\}$.
-

each local center $\mathbf{c}_i$ applies brute-force search for its m nearest neighbors $\mathbf{J}_{p,i}$ from the finer candidate set $\mathbf{S}_{p,i}$. The computational cost of brute search in $\mathbf{S}_{p,i}$ is negligible compared to $\mathcal{O}(n)$ due to $\alpha m \ll n$ and total block count $K \ll n$, thus filtered m -NNS achieves computational complexity $\mathcal{O}(\alpha m + K)$ and has a simple implementation. Fig. 3 expands Step 3 of Fig. 2 for one representative target block, from coarse candidate exchange and finer radius filtering through neighbor selection and GPU packing.

Fig. 3 makes the two filtering levels explicit for a target block $\mathbf{B}_{p,i}$. The coarse stage exchanges candidate blocks whose centers pass the radius test; the finer stage examines their points within λ to form $\mathbf{S}_{p,i}$, from which the m nearest neighbors $\mathbf{J}_{p,i}$ are selected. The resulting block, neighbor, observation, and size arrays are then packed as resident GPU inputs for the batched likelihood calculation. The brute-force selection is exact within $\mathbf{S}_{p,i}$; recovery of the global top- m neighbors depends on λ , whose sensitivity is quantified in Table V.

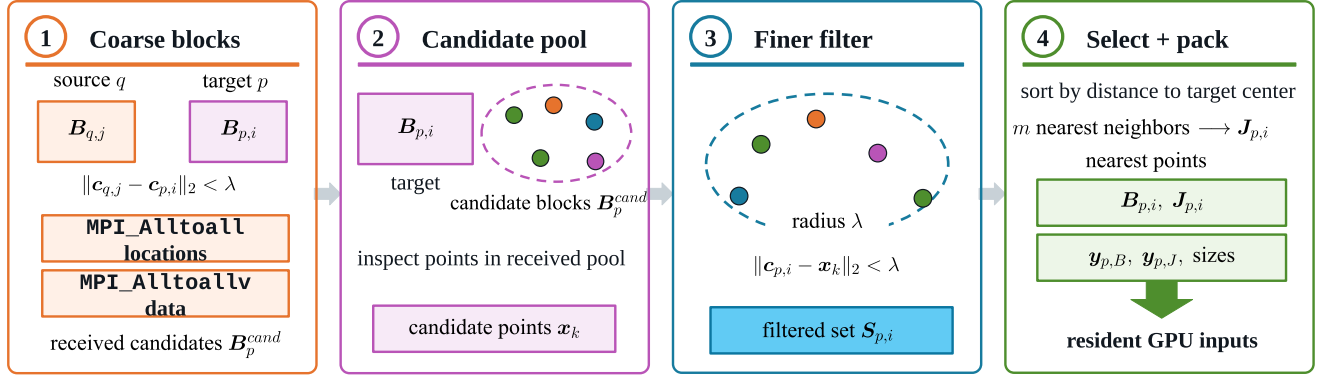


Fig. 3. Filtered NNS workflow for one representative target block $B_{p,i}$, from coarse candidate exchange and finer radius filtering through neighbor selection and GPU packing. Geometry and point counts are schematic.

D. Batched Log-likelihood Computations

This stage addresses repeated variable-size likelihood evaluation (C4).

While MPI enables scalable distributed computation across nodes, GPU acceleration delivers high single-node throughput for the SBV algorithm's log-likelihood computations. These computationally intensive phases involve batched dense factorizations and batched matrix multiplications resulting from block conditioning. These operations are entirely offloaded to the GPU using the MAGMA batched linear algebra library. Each conditioning block in the SBV algorithm is treated as an independent batch element, which exposes fine-grained parallelism across available accelerators. Batched Cholesky factorizations and GEMM updates are executed concurrently to ensure high device occupancy. Block sizes are chosen to balance kernel launch overhead, arithmetic intensity, and memory reuse. All GPU kernels operate in double precision (FP64) to maintain numerical stability and consistency with likelihood-based Gaussian process inference. Future work will investigate the extension to lower- or mixed-precision formats.

The SBV implementation is designed to utilize the GPU memory hierarchy efficiently. After partition construction on the CPU, all associated matrices are transferred to the high-bandwidth memory (HBM) and remain resident throughout the likelihood evaluation phase. The MAGMA library is used for batched linear algebra operations [50]. Conditioning blocks are reused across multiple batched operations, thereby increasing arithmetic intensity and reducing redundant memory accesses. The batched linear algebra kernels in SBV demonstrate high arithmetic intensity as a result of repeated reuse of conditioning matrices within each partition. This characteristic makes the workload well-suited for GPU acceleration.

Host-to-device transfers are performed only during the initial data staging following partition construction. No intermediate CPU-GPU synchronization is necessary during the batched factorization and matrix update phases. Device-to-host communication is restricted to the final likelihood values and gradient quantities. Consequently, host-device transfers do not occur within the primary iterative GPU kernels and account for only

a small portion of the total runtime. Maintaining data residency on the GPU throughout the computational phase minimizes PCIe/NVLink traffic. The large block/neighbor arrays are transferred once after preprocessing and remain resident; only the small parameter/range vector is copied to the device for each likelihood evaluation.

To explain this in detail, once data preprocessing is completed (Algorithm 4), the clusters $B_{1:P}$, their corresponding nearest-neighbor sets $J_{1:P}$, and the associated observations are transferred to the GPU global memory for subsequent log-likelihood optimization. Algorithm 5 outlines the computation of a single log-likelihood iteration, which must be evaluated multiple times until the prespecified optimization configuration is reached. Here, the batched operations are executed to compute the log-likelihood. This involves constructing local covariance matrices Σ^{lk} , Σ^{cross} and Σ^{con} using the scaled kernel K_θ . Subsequently, the GPU data structure is constructed in Step 4 in Fig. 2. Subsequently, batched Cholesky decompositions (POTRF) is utilized to factorize neighbor covariance matrices, Σ^{con} . Conditional mean and covariance updates are computed via a batched triangular solver (TRSM) and matrix multiplications (GEMM), while a batched triangular solver (TRSV) and determinant calculations derive per-block log-likelihoods $\ell_{p,i}$. Each node aggregates its local computations $\ell_p = \sum_{i=1}^{k_p} \ell_{p,i}$, and a global reduction across all nodes yields the final log-likelihood $\ell = \sum_{p=1}^P \ell_p$ using MPI_Allreduce.

E. GP Prediction

During the prediction stage, the pipeline follows the estimation process, with the key difference that, in Step (2) of Algorithm 5, GP calculations are replaced by conditional simulations. Specifically, the variance vector, $\sigma = (\sigma_1, \sigma_2, \dots, \sigma_{n^*})^\top$ is subtracted from the diagonal elements of the covariance matrices Σ^{new} , and μ^{new} serves as the predicted values, $y^* = (y_1^*, y_2^*, \dots, y_{n^*}^*)^\top$. Next, conditional simulations are performed by drawing 1000 samples from the normal distribution $\mathcal{N}(y_j^*, \sigma_j)$. Then, the sample mean $\tilde{\mu}_j$ and variance $\tilde{\sigma}_j^2$ are computed, and the 95% confidence interval is

Algorithm	5:	Batched	Block	Vecchia
$\ell_{lh}(\mathbf{B}_p, \mathbf{J}_p, \mathbf{y}_{p,B}, \mathbf{y}_{p,J})$.				
1:	Input:	In p th node, $\mathbf{B}_p$ cluster set, $\mathbf{J}_p$ nearest neighbor set, $\mathbf{y}_{p,B}$ and $\mathbf{y}_{p,J}$ observation sets.		
2:	Output:	log-likelihood ℓ_p		
3:	Step 1: Conditional Mean and Covariance Update			
4:	$\Sigma^{lk} \leftarrow$	batched $K_\theta(\mathbf{B}_p, \mathbf{B}_p)$.		
5:	$\Sigma^{con} \leftarrow$	batched $K_\theta(\mathbf{J}_p, \mathbf{J}_p)$.		
6:	$\Sigma^{cross} \leftarrow$	batched $K_\theta(\mathbf{J}_p, \mathbf{B}_p)$.		
7:	$\mathbf{L} \leftarrow$	batched $POTRF(\Sigma^{con})$		
8:	$\Sigma'^{cross} \leftarrow$	batched $TRSM(\mathbf{L}, \Sigma^{cross})$		
9:	$\Sigma'^{cor} \leftarrow$	batched $TRSV(\mathbf{L}, \mathbf{y}_{p,J})$		
		batched $GEMM(transpose(\Sigma'^{cross}), \Sigma'^{cross})$		
10:	$\mu^{cor} \leftarrow$	batched $GEMV(transpose(\Sigma'^{cross}), \mathbf{y}_{p,J})$		
11:	$\Sigma^{new} \leftarrow$	$\Sigma^{con} - \Sigma^{cor}$		
12:	$\mu^{new} \leftarrow$	μ^{cor}		
13:	Step 2: GPs calculations			
14:	$\mathbf{L}' \leftarrow$	batched $POTRF(\Sigma^{new})$		
15:	$\mathbf{v} \leftarrow$	batched $TRSV(\mathbf{L}', \mathbf{y}_{p,B} - \mu^{new})$		
16:	$\mathbf{u} \leftarrow$	batched $DotProduct(transpose(\mathbf{v}), \mathbf{v})$		
17:	$\mathbf{d} \leftarrow$	$2 \times \log(\text{determinant}(\mathbf{L}'))$		
18:	$\ell_p \leftarrow$	$-\frac{1}{2} \mathbf{1}^T (\mathbf{u} + \mathbf{d})$		
19:	return log-likelihood ℓ_p			

given by $(\tilde{\mu}_j - z_{\alpha/2}\tilde{\sigma}_j, \tilde{\mu}_j + z_{\alpha/2}\tilde{\sigma}_j)$, where $\alpha = 0.05, j = 1, 2, \dots, n^*$, which following [29].

Overall, the workflow minimizes inter-node communication and computations by scaling and partitioning, RAC, and filtered m -NNS, while GPU-batched kernels exploit fine-grained parallelism for covariance and log-likelihood computations. Moreover, this algorithm incorporates anisotropic scaling for high-dimensional inputs to ensure accurate emulation for computer models. This framework enables scalable and efficient evaluation of log-likelihoods for large-scale GPs in high-dimensional settings.

VII. COMPLEXITY ANALYSIS

Herein, we analyze the memory and computational complexity of SV and SBV, focusing on the GPU stage, which dominates due to repeated iterations, unlike the one-time CPU preprocessing.

For SV, the memory footprint is $\mathcal{O}(nm^2 + nm)$, primarily due to the storage of n small covariance matrices $\Sigma_{1:n}$ and corresponding conditioning vectors $\mathbf{y}_{1:n}$. In contrast, SBV partitions the data into bc blocks. Each block stores three covariance matrices, i.e., Σ_i^{lk} , Σ_i^{cross} , and Σ_i^{con} , and two observation vectors: $\mathbf{y}_{J_i}$ for the conditioning set and $\mathbf{y}_{B_i}$ for the block itself. Assuming an average block size of $bs = n/bc$, the memory complexity for each SBV component is: $\mathcal{O}(n \cdot bs)$ for Σ_i^{lk} , $\mathcal{O}(mn)$ for Σ_i^{cross} , $\mathcal{O}(bc \cdot m^2)$ for Σ_i^{con} , $\mathcal{O}(bc \cdot m)$ for $\mathbf{y}_{J_i}$, and $\mathcal{O}(n)$ for $\mathbf{y}_{B_i}$. Therefore, the total memory complexity is $\mathcal{O}(nm^2 + nm)$ for SV and $\mathcal{O}(n \cdot bs + mn + bc \cdot m^2)$ for SBV.

The dominant computational costs in SBV arise from Cholesky factorizations and matrix-matrix multiplications. The

TABLE II
COMPLEXITY ANALYSIS FOR SV AND SBV UNDER $m = 4 \times bs$

Method	Memory	Computational
SV	$\mathcal{O}(nm^2)$	$\mathcal{O}(nm^3)$
SBV	$\mathcal{O}(nm)$	$\mathcal{O}(nm^2)$

Cholesky operations contribute $\mathcal{O}(bc \cdot bs^3)$ for block matrices and $\mathcal{O}(bc \cdot m^3)$ for the conditioning sets. Matrix-matrix multiplications add a complexity of $\mathcal{O}(mn \cdot bs)$. Thus, the total computational complexity for SBV is $\mathcal{O}(n \cdot bs^2 + bc \cdot m^3 + mn \cdot bs)$, compared to $\mathcal{O}(nm^3)$ for SV, which performs a Cholesky factorization for each data point.

For high-dimensional input spaces, we recommend setting the number of nearest neighbors to four times the block size, i.e., $m = 4 \times bs$. Under this setting, the complexities are simplified and summarized in Table II. SBV achieves lower memory and computational costs than SV, especially when leveraging GPU-batched operations for large-scale GP models.

VIII. SIMULATION AND BENCHMARK EVALUATION

This section begins with a synthetic GP simulation using the Matérn kernel in high-dimensional input space, highlighting the critical role of scaling and clustering. We then apply the proposed SBV algorithm to the satellite drag benchmark dataset to demonstrate its effectiveness and accuracy in emulating complex computer models.

A. Synthetic Data Simulation

The simulation framework follows the design outlined in [28], ensuring consistency for comparative analysis. The synthetic dataset is generated from a zero-mean GP with a Matérn kernel ($\nu = 3.5$) using Equation (6). The input space is $\mathbf{x} \in [0, 1]^{10}$. The response is modeled as $y(\mathbf{x}) \sim \mathcal{N}(0, K_\theta(\mathbf{x}, \mathbf{x}'))$, where the parameter vector is given by $\theta = (\sigma^2, \tau^2, \beta)$. The true parameters are set as $(\sigma^2, \tau^2) = (1.0, 0)$, with range parameters $\beta_1 = \beta_2 = 0.05$ for the relevant dimensions and $\beta_3 = \beta_4 = \dots = \beta_{10} = 5$ for the irrelevant ones. We evaluate four Vecchia-based GPs on this synthetic dataset: CV [16], BV [29], SV [28], and our proposed SBV. The choice of $bs_{est} = bs_{pred} = 10$ in BV and SBV reflects a balance between computational efficiency and predictive accuracy. Model performance is assessed using both the KL divergence defined in Equation (4) and the Mean Squared Prediction Error (MSPE). To isolate approximation error and avoid conflating it with parameter estimation error, the true parameters $(\sigma^2, \tau^2, \beta)$ are directly supplied to all four models.

The results are presented in Fig. 4. In Fig. 4(a), the proposed SBV method achieves the lowest KL divergence, followed by SV, BV, and CV, highlighting the advantage of combining input scaling and clustering over the CV approach. The performance gains of SBV arise from its ability to address input anisotropy through joint scaling and to increase neighborhood coverage through clustering. Fig. 4(b) shows a similar trend regarding predictive accuracy, where SBV again yields the lowest MSPE,

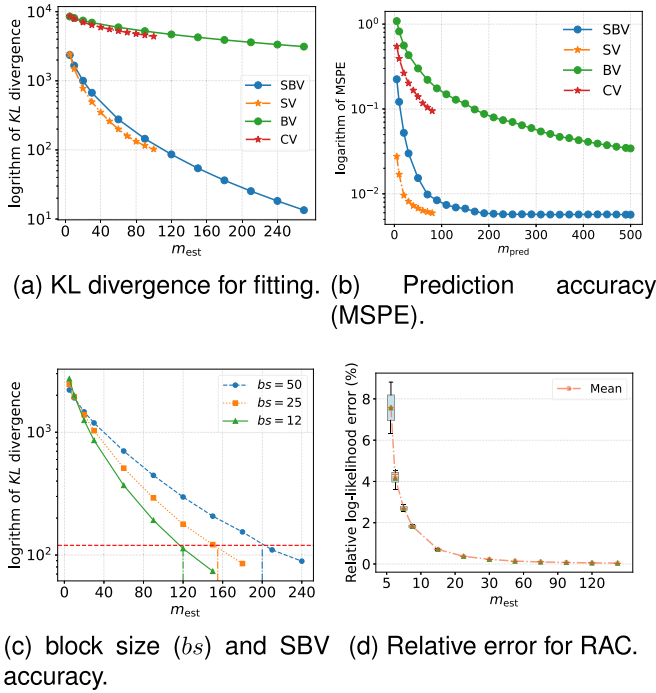


Fig. 4. Comparison of Vecchia-based GP methods on model fitting and prediction accuracy, i.e., Classic Vecchia (CV), Block Vecchia (BV), Scaled Vecchia (SV), Scaled Block Vecchia (SBV). Subfigures (a) and (b) evaluate the GP variants fitting (KL divergence) and prediction separately, showing that scaling and block conditioning enable better accuracy; subfigure (c) investigates SBV with different block sizes (bs), showing that a larger m and a smaller block size yield the best fitting accuracy; subfigure (d) clarifies that the proposed RAC has comparable and robust log-likelihood with the K-means clustering and has the advantage of computationally linear complexity.

followed by SV, BV, and CV. These results are consistent with the findings on the KL divergence and further validate the benefits of incorporating input scaling and structured clustering. In Fig. 4(c), we examine the impact of block size on KL divergence. Smaller blocks ($bs_{est} = 12$) yield lower KL divergence than larger ones ($bs_{est} = 50$), indicating improved approximation accuracy with finer partitioning. However, larger blocks impose greater computational overhead, thereby enabling more effective GPU utilization. Therefore, the choice of block size bs introduces a trade-off between accuracy and computational cost. Fig. 4(d) illustrates the relative error and its variance of the log-likelihood computed using two clustering methods: K-means and the proposed RAC approach. To assess the robustness of RAC to anchor randomness, we perform five independent random anchor selections and report the mean error and its variance. The results show that both the approximation error and its variability decrease as the conditioning size m increases, eventually becoming nearly negligible. This trend confirms the stability of RAC for sufficiently large m . Furthermore, larger conditioning sizes are typically required in large-scale and high-dimensional applications to maintain high approximation accuracy.

B. Evaluation on Satellite Drag Benchmark

The satellite drag dataset in [30] is a widely used benchmark for evaluating GP-based models in high-dimensional settings

TABLE III
VECCHIA-BASED GP CONFIGURATIONS ON THE SATELLITE DRAG DATASET, SHOWING BLOCK SIZES (bs_{est} , bs_{pred}) AND NEIGHBOR COUNTS (m_{est} , m_{pred}) FOR ESTIMATION AND PREDICTION

Model	SV	SBV ₁	SBV ₂	SBV ₃	SBV ₄	SBV ₅	SBV ₆
bs_{est}	1	100	100	100	100	100	100
bs_{pred}	1	5	5	5	5	5	5
m_{est}	50	200	200	200	400	400	400
m_{pred}	140	200	400	600	200	400	600

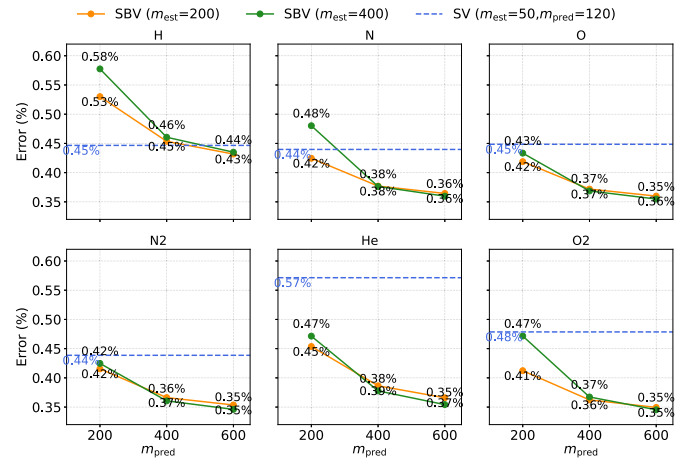


Fig. 5. Root Mean Squared Percentage Error (RMSPE) of Vecchia-based GP models on six outputs from the satellite drag benchmark.

[28], [30]. It is generated from a high-fidelity simulator that models atmospheric drag coefficients in low Earth orbit (LEO). The dataset comprises 2M simulation runs for each of six primary atmospheric species: atomic oxygen (O), molecular oxygen (O₂), atomic nitrogen (N), molecular nitrogen (N₂), helium (He), and atomic hydrogen (H). The total drag coefficient is computed as a weighted average of these species. Each species-specific dataset has an 8-dimensional input space: relative velocity, surface temperature, atmospheric temperature, yaw angle, pitch angle, and two accommodation coefficients.

Following the experimental design in [28], we split the dataset into 90% for training and 10% for testing and performed 10-fold cross-validation for robust evaluation. We adopt the same Matérn kernel with nugget effects and smoothness parameter $\nu = 3.5$ as in [28]. Model performance is assessed using the Root Mean Squared Percentage Error (RMSPE) and the estimated kernel parameters. The models compared in this study are mentioned in Table III.

For model fitting, SV uses $m_{est} = 50$ nearest neighbors on a 50K-sample subset, following the configuration in [28], which reported only marginal gains in predictive accuracy beyond this setting, given the increased computational cost. In contrast, SBV leverages the full dataset with larger neighbor counts ($m_{est} = 200$ or 400) while maintaining a consistent average block size of $bs_{est} = 100$ across all configurations. Due to the high memory and computational demands of SV, fitting the full dataset on a single GPU is infeasible, making SBV a more practical and scalable alternative.

Fig. 5 presents the RMSPE results for Vecchia-based GP models on the benchmark dataset. SBV consistently achieves the

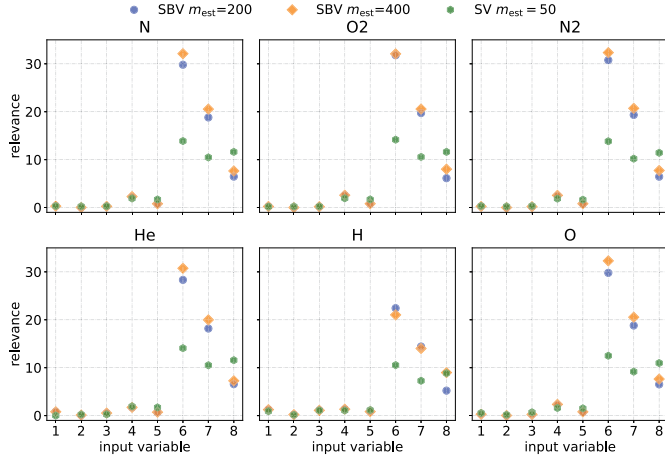


Fig. 6. Estimated input relevance ($1/\beta_i$) for SV and SBV on the satellite drag benchmark.

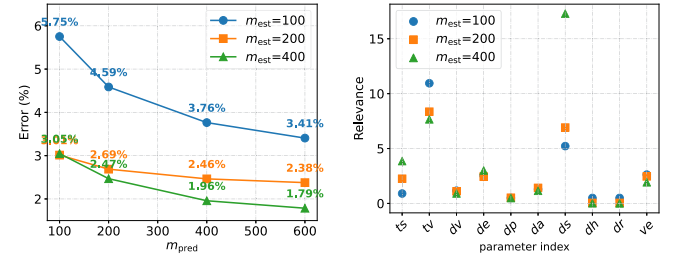
lowest RMSPE, demonstrating the effectiveness of combining input scaling with clustering. Increasing the number of prediction neighbors m_{pred} improves accuracy; for example, SBV with $m_{\text{pred}} = 400$ reduces RMSPE by approximately 15% on average compared to $m_{\text{pred}} = 200$. To better understand the improved accuracy achieved by SBV, Fig. 6 compares the estimated inverse lengthscales ($1/\beta_i$) for SV and SBV. Both methods identify the final three input dimensions as most relevant while treating the others as less influential. However, the parameter estimates from SBV differ significantly from those from SV, underscoring the importance of using a larger number of nearest neighbors for accurate GP fitting. From a computational and memory perspective, SV faces challenges in handling large neighbor counts because substantial increases in cost and memory usage limit efficient GPU utilization. In contrast, SBV remains scalable and efficient under these conditions.

C. MetaRVM Emulator

MetaRVM [31] is an R package that simulates the spread of respiratory viruses using a graph-based probabilistic model. By dividing a total population into sub-populations such as zones, age groups, or races, it models disease dynamics within and between these groups. This structure enables the simulation of interactions and disease spread within and between subpopulations. In healthcare, such models are crucial for understanding transmission patterns, evaluating intervention strategies, and informing public health policies. They help predict outbreak trajectories, assess the potential impact of vaccinations, and optimize resource allocation, ultimately aiding in the control and prevention of respiratory infections. The parameters in the MetaRVM simulator are listed in Table IV. To understand the relationship between the input parameters and the accumulated number of hospitalizations (output) over 100 days in one population, we randomly choose 50M sets of input variables and obtain the accumulated hospitalizations using the MetaRVM simulator. The 50M outputs are then divided into 90% for fitting SBV GPs with a Matérn kernel at $\nu = 3.5$ and 10% for prediction, where the RMSPE is reported. The input is also scaled to $[0, 1]$, and

TABLE IV
THE MetaRVM DATASET SIMULATION PARAMETERS

Input	Meaning	Bound
ts	transmissibility for susceptible	(0.1, 0.9)
tv	transmissibility for vaccinated	(0.1, 0.9)
dv	mean duration in vaccinated state	(30, 90)
de	mean duration in exposed state	(1, 5)
dp	mean duration in infectious presymptomatic state	(1, 3)
da	mean duration in infectious asymptomatic state	(1, 9)
ds	mean duration in infectious symptomatic state	(1, 9)
dh	mean duration in hospitalized state	(1, 5)
dr	mean duration in recovered state	(30, 90)
ve	vaccine efficacy	(0.3, 0.8)



(a) RMSPE.

(b) Estimated parameters.

Fig. 7. Root Mean Squared Percentage Error (RMSPE) and estimated relevance of SBV algorithm on MetaRVM dataset. (a) RMSPE. (b) Estimated parameters.

the output is normalized with a mean of 1 to avoid abnormal values in RMSPE. In the SBV, we set $bs_{\text{est}} = 100$, $bs_{\text{pred}} = 25$ and investigate accuracy as a function of the number of nearest neighbors.

The results are presented in Fig. 7, where a large m for both estimation and prediction generally leads to improved RMSPE. In the parameter estimation, the relevance dh and dr are close to 0, which aligns with our expectation, as these parameters are not involved in the accumulated number of hospitalizations in the MetaRVM simulator. Besides, for ds and tv , the estimation varies significantly with the increasing value of m_{est} .

D. Radius-Filter Sensitivity

To isolate the effect of the radius without rerunning distributed SBV, we timed a controlled two-dimensional radius-filter microbenchmark ($n = 200,000$, 400 queries, $m = 100$; five repetitions), sweeping $\lambda = s\lambda_0$, where $\lambda_0 = 1.2\sqrt{m/(\pi n)}$. Points and queries cover grid lookup, radius filtering, and partial selection. This experiment does not measure block construction, MPI communication, or total SBV runtime, and λ_0 is a study-specific reference rather than the deployed setting. Here, s is the radius multiplier, candidates/ m is the mean retained-candidate count divided by m , and recall at m is the fraction of exact top- m neighbors recovered; see Table V.

Increasing λ raises candidate work and filtering time; in this controlled study, recall saturates by $1.5\lambda_0$, after which a larger radius adds cost without improving recall. In concrete terms, moving from $0.75\lambda_0$ to λ_0 raises candidates/ m from 0.79 to 1.41 and recall at m from 79.23% to 99.68%. Both $1.5\lambda_0$ and $2\lambda_0$ attain 100% recall, but their median times increase from 8.79 to

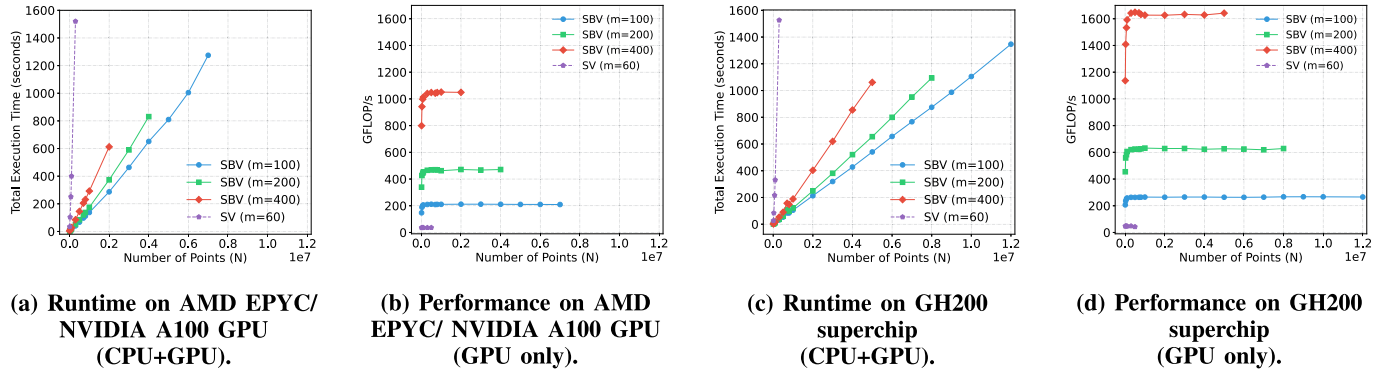


Fig. 8. Performance comparison of SBV and SV methods with 500 MLE iterations on a single AMD EPYC with NVIDIA A100 (40 GB) and a single NVIDIA GH200 superchip (96 GB). (a) Runtime on AMD EPYC/ NVIDIA A100 GPU (CPU+GPU). (b) Performance on AMD EPYC/ NVIDIA A100 GPU (GPU only). (c) Runtime on GH200 superchip (CPU+GPU). (d) Performance on GH200 superchip (GPU only).

TABLE V
STANDALONE CPU RADIUS-FILTER SENSITIVITY. TIMES ARE FIVE-RUN
MEDIAN FOR ALL 400 QUERIES

λ/λ_0	candidates/ m	recall at m	time (ms)
0.75	0.79	79.23%	4.71
1.00	1.41	99.68%	5.96
1.25	2.19	99.97%	7.40
1.50	3.15	100.00%	8.79
2.00	5.57	100.00%	11.97

11.97 ms, respectively, illustrating how the radius controls both neighbor coverage and filtering work in this standalone study.

VIII. PERFORMANCE AND ENERGY EVALUATION

We conduct experiments on two modern NVIDIA GPU architectures, A100 and GH200, hosted at the Jülich Supercomputing Centre (JSC) on the JURECA-DC and JUPITER systems including the early access JUPITER development system JEDI. JUPITER features about 6000 nodes, each equipped with four NVIDIA GH200 Grace Hopper Superchips. 48 of these nodes were accessible early as the JEDI system. Each Superchip integrates a 72-core Grace CPU (3.1 GHz, 120 GB) and a Hopper GPU (96 GB HBM3), connected via NVLink-C2C (900 GB/s), with a total power consumption of approximately 680 W per Superchip. JURECA-DC contains 192 GPU-equipped nodes, each with two 64-core AMD EPYC 7742 CPUs (2.25 GHz, 512 GB, 225 W per socket) and four NVIDIA A100 GPUs (40 GB HBM2, 400 W per GPU). Both systems use InfiniBand interconnects: JUPITER and JEDI provide four NDR200 (200 Gbit/s) links per node, while JURECA-DC GPU offers two HDR (200 Gbit/s) links per node. Our framework is compiled with GCC 13.3.0, uses CUDA 11.8/12.0, and links against MAGMA 2.7.2 [32], [33] and NLOpt 2.7.1. All experiments run in double precision and are repeated for consistency.

All following experiments set the default $bs = 100$, 10-dimensional input $[0, 1]^{10}$, and only consider the estimation stage, which is the most time-consuming part in the whole lifetime of emulations; thus, m represents the number of nearest neighbors for estimation and uses Maximum Likelihood Estimation (MLE). Here, we set the number of iterations to 500, as

the average number of optimization rounds across benchmark datasets is 500.

A. Single-Node Performance

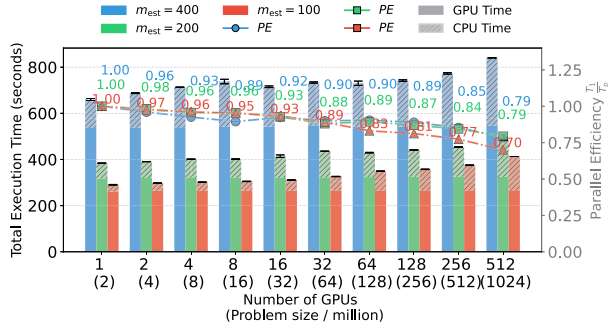
Fig. 8 illustrates the runtime and GPU throughput performance of the SBV and SV methods on two nodes: an NVIDIA A100 with its host CPU, and the GH200 Superchip. Subfigures (a) and (c) show that SBV consistently outperforms SV in total execution time across different values of m , demonstrating superior scalability as the dataset size increases. As expected, larger values of m increase the total runtime due to the higher computational cost per iteration; however, they also yield significantly better approximation and prediction accuracy, as shown in the accuracy section.

Subfigures (b) and (d) highlight the GPU throughput (in GFLOP/s) for a single iteration, showing that SBV achieves significantly higher sustained performance than SV on both platforms. Notably, the GH200 delivers performance exceeding 1600 GFLOP/s with $m = 400$. These results confirm that SBV is faster and more GPU-efficient than SV, particularly when utilizing larger neighborhoods on modern architectures, such as the GH200.

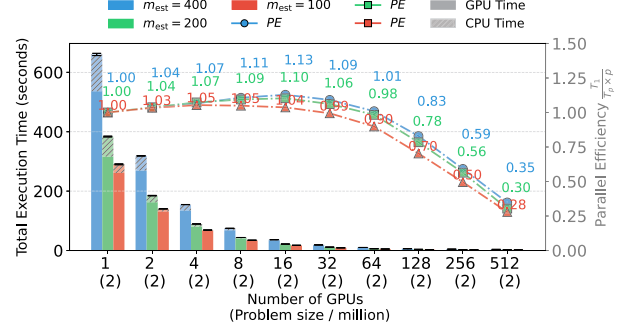
While the GH200 offers substantially higher theoretical FP64 peak performance compared to the A100, the SBV workload is not exclusively compute-bound. The primary kernels involve batched Cholesky factorizations and moderate-sized GEMM/TRSM operations on blocks with $bs \approx 100$ and $m \leq 400$. For these matrix dimensions, arithmetic intensity remains moderate, and performance is limited in part by memory bandwidth and batched-kernel occupancy rather than by peak FP64 throughput. As a result, the observed performance gains align more closely with increases in effective HBM bandwidth than with theoretical peak FLOP ratios. This pattern suggests that SBV operates within a mixed memory-and-compute regime, rather than failing to utilize the capabilities of the newer architecture.

B. Scalability Across Multiple Nodes

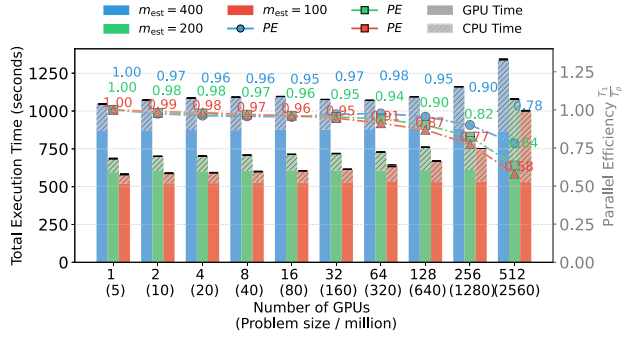
Fig. 9 demonstrates the weak and strong scaling performance of the distributed SBV implementation on NVIDIA A100 and



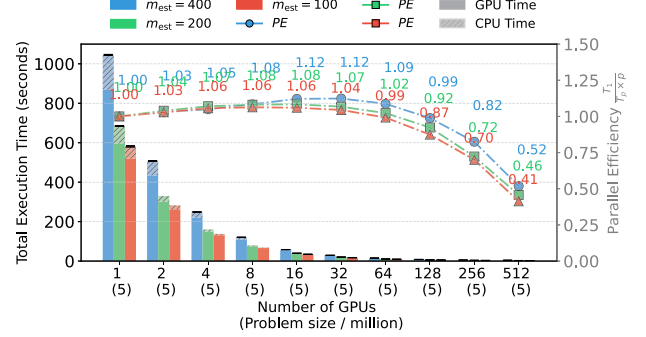
(a) Weak scalability and PE (A100) (2M–1024M points).



(b) Strong Scalability and PE (A100) (2M points).



(c) Weak scalability and PE (GH200) (5M–2.56B points).



(d) Strong Scalability and PE (GH200) (5M points).

Fig. 9. A100/GH200 weak and strong scaling on up to 512 GPUs, with each run capped at a maximum of 500 likelihood-objective evaluations. The largest reported weak-scaling workload has a nominal input count of 2.56 B on 512 GH200 GPUs, demonstrating execution of the capped likelihood-evaluation workload at that scale and indicating potential for extremely large-scale applications; it does not establish optimizer convergence, held-out prediction, or end-to-end emulation at 2.56 B inputs. Here p denotes the number of GPUs. For weak scaling, $PE_{\text{weak}}(p) = T_1/T_p$, where T_1 is the base-problem runtime on one GPU and T_p is the runtime for the p -times larger nominal problem on p GPUs. For strong scaling, $PE_{\text{strong}}(p) = T_1/(pT_p)$, where T_1 and T_p refer to the same fixed problem. (a) Weak scalability and PE (A100) (2M–1024M points). (b) Strong Scalability and PE (A100) (2M points). (c) Weak scalability and PE (GH200) (5M–2.56 B points). (d) Strong Scalability and PE (GH200) (5M points).

GH200 architectures, scaling up to 512 GPUs. In weak scaling (Subfigures (a) and (c)), both systems maintain high parallel efficiency (PE) as the problem size increases proportionally with the number of GPUs. The drops at larger scales are derived from the CPU computation, specifically Step 2 in Algorithm 4, which can be reduced using a smaller λ in Eq. 7. Specifically, the candidate count does not scale proportionally and exhibits some stochastic variability, leading to non-uniform load balancing across tasks. A gradually decreasing λ is recommended in the NNS as the number of GPUs increases.

In strong scaling (Subfigures (b) and (d)), total execution time decreases nearly linearly with increasing GPU count. Parallel efficiency exceeds ideal scaling ($PE > 1.0$) in several cases due to fewer nearest-neighbor candidates in Algorithm 4, where λ can be increased accordingly to ensure there are enough candidates for NNS. The drops in the larger GPU count originate from the overhead of multiple function calls, where up to 500 likelihood-objective evaluations are executed. Their accumulation cannot be hidden as the computational task is light for each GPU, where each evaluation only costs around 30 milliseconds. The largest GH200 weak-scaling run uses an input count of 2.56 B on 512 GPUs and measures 500 likelihood-objective evaluations. It demonstrates the implementation's ability to execute SBV likelihood computations at this scale and its potential for extremely large-scale applications.

In both scaling experiments, the influence of the neighborhood size m_{est} is also evident: larger values, such as $m = 400$, increase the computational workload per point and runtime, but yield higher GPU throughput and overall hardware utilization. Smaller values, such as $m = 100$, result in faster runtimes but less efficient GPU utilization, especially on the GH200. Mid-range values (e.g., $m = 200$) balance performance and efficiency.

A detailed breakdown shows that GPU compute time dominates execution, accounting for over 80% of the total runtime in large problem sizes. However, the remaining CPU-side computation introduces variations in scaling. On A100, limited CPU resources (32 cores per task) lead to a gradual drop in PE , whereas the 72-core CPU on GH200 better hides this overhead. This difference arises from the CPU's role in maintaining a globally sorted array for block reordering and candidate filtering, as in Equation (2).

C. Power Consumption Analysis

Fig. 10 presents the power consumption profiles of a single A100 and GH200 superchip GPU while running 500 log-likelihood iterations across varying neighborhood sizes (m_{est}), using a problem size of 2M points on A100 and 5M on GH200, the largest size that fits in GPU memory. As expected, both power

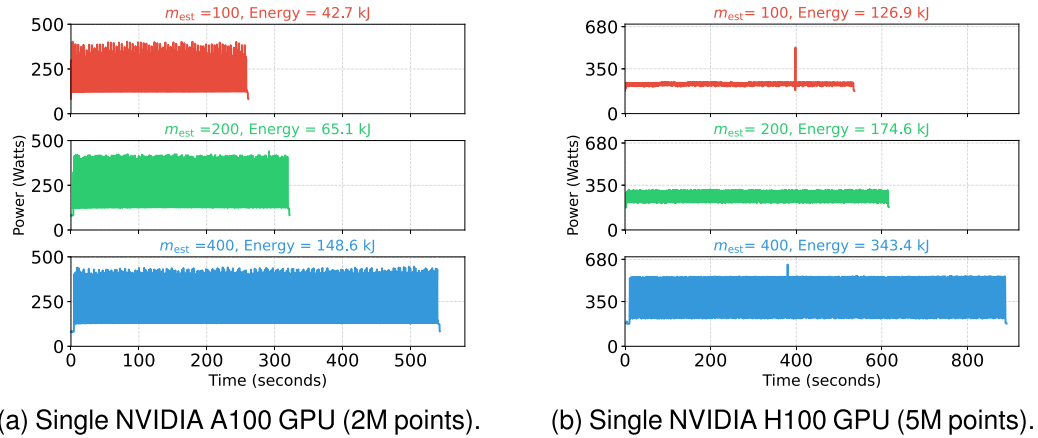


Fig. 10. Power consumption/energy (kJ) on a single GPU for two NVIDIA GPUs over 500 iterations per m_{est} . A100 GPU with 400 W max power cap and Hopper GPU with (680 W - the power usage of the Grace CPU, RAM, and system I/O) max power cap. (a) Single NVIDIA A100 GPU (2M points). (b) Single NVIDIA H100 GPU (5M points).

draw and total energy consumption increase with larger m_{est} due to the higher computational workload. On A100 (Fig. 10(a)), power remains well below the 500 W limit, with total energy ranging from 42.7 kJ ($m_{est} = 100$) to 148.6 kJ ($m_{est} = 400$). In contrast, the Hopper GPUs as part of the GH200 superchips (Fig. 10(b)) consume significantly more energy, up to 343.4 kJ, while handling a larger problem size, enabled by a higher power envelope. For GH200 superchips, the maximum power consumption per superchip is 680 W (680 W for each GH200 in JEDI) and the Grace CPU is prioritized. To avoid power shortages and resulting performance drops in Hopper GPUs, the maximum CPU power consumption can be capped (including RAM and system I/O). On the JEDI system, the default CPU power cap is 100 W, leaving at least 580 W for the Hopper GPUs.

We also compare our results to the power profiles of exact GPs reported in [51], which measure the energy consumption of Cholesky-based FP64 and exact GP variants on V100, A100, and H100 GPUs. In that study, a single MLE iteration requires more than 140 kJ on A100 and 340 kJ on H100, even for relatively small matrix sizes (122,880 points). In contrast, our full SBV-based MLE estimation, which includes 500 iterations on much larger datasets (2M and 5M points), consumes only ~ 12 – 40% of the energy required for a single exact GP iteration on those systems, despite handling problems that are $16\times$ to $40\times$ larger. These results highlight the energy efficiency of SBV on modern GPU architectures while demonstrating its scalability for high-dimensional GP modeling.

IX. CONCLUSION

This paper introduced the Scaled Block Vecchia (SBV) approximation, a novel framework for scalable GP emulation tailored for high-dimensional datasets on distributed GPU systems. By integrating anisotropic input scaling with block-based conditioning and leveraging batched GPU kernels, SBV substantially reduces memory and computational costs while maintaining high predictive accuracy. To our knowledge, this is the first distributed implementation of a Vecchia-based GP

approximation. Through extensive experiments on synthetic and real-world data, including the satellite drag benchmark and the MetaRVM epidemiological model, we demonstrate scalability up to 512 GPUs and the ability to handle up to 2.56 B points, achieving energy efficiency and state-of-the-art accuracy.

ACKNOWLEDGMENTS

We gratefully acknowledge Ahmad Abdelfattah from the University of Tennessee, Knoxville (UTK), for his support and assistance in optimizing the use of the MAGMA library. We also thank the KAUST Supercomputing Laboratory (KSL) team for providing access to the Ibex cluster for initial runs and GPU experiments. We further acknowledge the computing time provided by the John von Neumann Institute for Computing (NIC) on the supercomputer JURECA-DC. This project received access to JEDI, a preparation system for the JUPITER supercomputer, and JUPITER through the JUPITER Research and Early Access Program (JUREAP). JUPITER is funded by the EuroHPC Joint Undertaking, the German Federal Ministry of Research, Technology, and Space, and the Ministry of Culture and Science of the German state of North Rhine-Westphalia.

REFERENCES

- [1] V. Eyring et al., "Overview of the coupled model intercomparison project phase 6 (CMIP6) experimental design and organization," *Geoscientific Model Develop.*, vol. 9, no. 5, pp. 1937–1958, 2016.
- [2] D. Kochkov et al., "Neural general circulation models for weather and climate," *Nature*, vol. 632, no. 8027, pp. 1060–1066, 2024.
- [3] T. Børvik, O. Hopperstad, T. Berstad, and M. Langseth, "A computational model of viscoplasticity and ductile damage for impact and penetration," *Eur. J. Mechan.-A/Solids*, vol. 20, no. 5, pp. 685–712, 2001.
- [4] D. J. Walters et al., "Bayesian calibration of strength parameters using hydrocode simulations of symmetric impact shock experiments of Al-5083," *J. Appl. Phys.*, vol. 124, no. 20, 2018, Art. no. 205105.
- [5] V. Springel, "The cosmological simulation code GADGET-2," *Monthly Notices Roy. Astron. Soc.*, vol. 364, no. 4, pp. 1105–1134, 2005.
- [6] A. S. Mancini, D. Piras, J. Alsing, B. Joachimi, and M. P. Hobson, "Cosmopower: Emulating cosmological power spectra for accelerated bayesian inference from next-generation surveys," *Monthly Notices Roy. Astron. Soc.*, vol. 511, no. 2, pp. 1771–1788, 2022.
- [7] E. Lawrence et al., "The Mira-Titan universe. II. matter power spectrum emulation," *Astrophysical J.*, vol. 847, no. 1, 2017, Art. no. 50.

- [8] S. Fiore, M. Bakhouya, and W. W. Smari, "On the road to exascale: Advances in high performance computing and simulation: An overview and editorial," in *Future Gener. Comput. Syst.*, vol. 82, pp. 450–458, 2018.
- [9] T. J. Santner, B. J. Williams, W. I. Notz, and B. J. Williams, *The Design and Analysis of Computer Experiments*, vol. 1. Berlin, Germany: Springer, 2003.
- [10] S. Abdulah et al., "Boosting earth system model outputs and saving petabytes in their storage using exascale climate emulators," in *Proc. SC24: Int. Conf. High Perform. Comput., Netw., Storage Anal.*, 2024, pp. 1–12.
- [11] I. Andrianakis and P. G. Challenor, "The effect of the nugget on Gaussian process emulators of computer models," *Comput. Statist. Data Anal.*, vol. 56, no. 12, pp. 4215–4228, 2012.
- [12] A. Saltelli, P. Annoni, I. Azzini, F. Campolongo, M. Ratto, and S. Tarantola, "Variance based sensitivity analysis of model output. design and estimator for the total sensitivity index," *Comput. Phys. Commun.*, vol. 181, no. 2, pp. 259–270, 2010.
- [13] M. A. Winkel, J. W. Stallrich, C. B. Storlie, and B. J. Reich, "Sequential optimization in locally important dimensions," *Technometrics*, vol. 63, no. 2, pp. 236–248, 2021.
- [14] F. M. A. Acosta, "Radial basis function and related models: An overview," *Signal Process.*, vol. 45, no. 1, pp. 37–58, 1995.
- [15] G. Arora, K. Bala, H. Emadifar, and M. Khademi, "A review of radial basis function with applications explored," *J. Egyptian Math. Soc.*, vol. 31, no. 1, pp. 1–14, 2023.
- [16] A. V. Vecchia, "Estimation and model identification for continuous spatial processes," *J. Roy. Stat. Soc. Ser. B, Stat. Methodol.*, vol. 50, no. 2, pp. 297–312, 1988.
- [17] M. Katzfuss and J. Guinness, "A general framework for vecchia approximations of Gaussian processes," *Stat. Sci.*, vol. 36, no. 1, pp. 124–141, 2021.
- [18] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Exa-GeoStat: A high performance unified software for geostatistics on many-core systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 29, no. 12, pp. 2771–2784, Dec. 2018.
- [19] S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Geostatistical modeling and prediction using mixed precision tile Cholesky factorization," in *Proc. IEEE 26th Int. Conf. High Perform. Comput., Data, Analytics*, 2019, pp. 152–162.
- [20] S. Abdulah et al., "Accelerating geostatistical modeling and prediction with mixed-precision computations: A high-productivity approach with parsec," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 4, pp. 964–976, Apr. 2022.
- [21] R. Furrer, M. G. Genton, and D. Nychka, "Covariance tapering for interpolation of large spatial datasets," *J. Comput. Graphical Statist.*, vol. 15, no. 3, pp. 502–523, 2006.
- [22] M. Bevilacqua, A. Fassò, C. Gaetan, E. Porcu, and D. Velandia, "Covariance tapering for multivariate Gaussian random fields estimation," *Stat. Methods Appl.*, vol. 25, pp. 21–37, 2016.
- [23] J. Shi, S. Abdulah, Y. Sun, and M. G. Genton, "Fully decentralized inference for spatial data using low-rank models," *J. Royal Statistical Soc. Series B: Statistical Methodology*, 2026, doi: [10.1093/jrsssb/qkag113](https://doi.org/10.1093/jrsssb/qkag113).
- [24] 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 *Proc. IEEE Int. Conf. Cluster Comput.*, 2018, pp. 98–108.
- [25] M. L. O. Salvaña, S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Parallel space-time likelihood optimization for air pollution prediction on large-scale systems," in *Proc. Platform Adv. Sci. Comput. Conf.*, 2022, pp. 1–11.
- [26] S. Mondal, S. Abdulah, H. Ltaief, Y. Sun, M. G. Genton, and D. E. Keyes, "Tile low-rank approximations of non-Gaussian space and space-time tukey g-and-h random field likelihoods and predictions on large-scale systems," *J. Parallel Distrib. Comput.*, vol. 180, 2023, Art. no. 104715.
- [27] Q. Pan, S. Abdulah, M. G. Genton, D. E. Keyes, H. Ltaief, and Y. Sun, "GPU-accelerated Vecchia approximations of Gaussian processes for geospatial data using batched matrix computations," in *Proc. 39th Int. Conf. ISC High Perform. Res. Paper*, 2024, pp. 1–12.
- [28] M. Katzfuss, J. Guinness, and E. Lawrence, "Scaled vecchia approximation for fast computer-model emulation," *SIAM/ASA J. Uncertainty Quantification*, vol. 10, no. 2, pp. 537–554, 2022.
- [29] Q. Pan, S. Abdulah, M. G. Genton, and Y. Sun, "Block Vecchia Approximation for Scalable and Efficient Gaussian Process Computations," *Technometrics*, vol. 67, no. 3, pp. 1–18, 2025.
- [30] F. Sun, R. B. Gramacy, B. Haaland, E. Lawrence, and A. Walker, "Emulating satellite drag from large simulation experiments," *SIAM/ASA J. Uncertainty Quantification*, vol. 7, no. 2, pp. 720–759, 2019.
- [31] A. Fadikar et al., "Developing and deploying a use-inspired metapopulation modeling framework for detailed tracking of stratified health outcomes," in *Proc. 2025 Winter Simulation Conf.*, 2025, pp. 734–745.
- [32] A. Abdelfattah et al., "A set of batched basic linear algebra subprograms and lapack routines," *ACM Trans. Math. Softw.*, vol. 47, no. 3, pp. 1–23, 2021.
- [33] E. Agullo et al., "Numerical linear algebra on emerging architectures: The PLASMA and MAGMA projects," in *J. Phys.: Conf. Ser.*, vol. 180, no. 1. 2009, Art. no. 012037.
- [34] A. Fadikar, D. Higdon, J. Chen, B. Lewis, S. Venkatramanan, and M. Marathe, "Calibrating a stochastic, agent-based model using quantile-based emulation," *SIAM/ASA J. Uncertainty Quantification*, vol. 6, no. 4, pp. 1685–1706, 2018.
- [35] J. Guinness, "Permutation and grouping methods for sharpening Gaussian process approximations," *Technometrics*, vol. 60, no. 4, pp. 415–429, 2018.
- [36] A. Sauer, A. Cooper, and R. B. Gramacy, "Vecchia-approximated deep gaussian processes for computer experiments," *J. Comput. Graphical Statist.*, vol. 32, no. 3, pp. 824–837, 2023.
- [37] P. Kündig and F. Sigrist, "Iterative methods for Vecchia-Laplace approximations for latent Gaussian process models," *J. Amer. Stat. Assoc.*, vol. 120, no. 550, pp. 1267–1280, 2025.
- [38] J. Cao, J. Guinness, M. G. Genton, and M. Katzfuss, "Scalable Gaussian-process regression and variable selection using Vecchia approximations," *J. Mach. Learn. Res.*, vol. 23, no. 348, pp. 1–30, 2022.
- [39] F. Jimenez and M. Katzfuss, "Scalable Bayesian optimization using Vecchia approximations of Gaussian processes," in *Proc. Int. Conf. Artif. Intell. Statist.*, 2023, pp. 1492–1512.
- [40] R. Huser, M. L. Stein, and P. Zhong, "Vecchia likelihood approximation for accurate and fast inference with intractable spatial max-stable models," *J. Comput. Graphical Statist.*, vol. 33, no. 3, pp. 978–990, 2024.
- [41] J. Cao, and M. Katzfuss, "Linear-Cost Vecchia Approximation of Multivariate Normal Probabilities," *J. Amer. Statistical Assoc.*, vol. 121, no. 553, pp. 1–14, 2025.
- [42] C. E. Rasmussen and C. K. Williams, *Gaussian Processes for Machine Learning*. Cambridge, MA, USA: MIT Press, 2006.
- [43] T. M. Cover and J. A. Thomas, *Elements of Information Theory*. Hoboken, NJ, USA: Wiley, 2006.
- [44] K. P. Murphy, *Machine Learning: A Probabilistic Perspective*. Cambridge, MA, USA: MIT Press, 2012.
- [45] M. G. Genton, "Classes of kernels for machine learning: A statistics perspective," *J. Mach. Learn. Res.*, vol. 2, pp. 299–312, Dec. 2001.
- [46] S. G. Johnson, "The NLOpt nonlinear-optimization package," 2007. [Online]. Available: <https://github.com/stevengj/nlopt>.
- [47] M. R. Abbasifard, B. Ghahremani, and H. Naderi, "A survey on nearest neighbor search methods," *Int. J. Comput. Appl.*, vol. 95, no. 25, pp. 39–52, 2014.
- [48] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. El Abbadi, "Constrained nearest neighbor queries," in *Proc. Int. Symp. Spatial Temporal Databases*, 2001, pp. 257–276.
- [49] K. L. Clarkson et al., "Nearest-neighbor searching and metric space dimensions," in *Nearest-Neighbor Methods in Learning and Vision: Theory and Practice*. Cambridge, MA, USA: MIT Press, 2006, pp. 15–59.
- [50] S. Tomov, J. Dongarra, V. Volkov, and J. Demmel, *Magma Library*. Univ. Tennessee Univ. California, Knoxville, TN, Berkeley, CA.
- [51] Q. Cao, S. Abdulah, H. Ltaief, M. G. Genton, D. Keyes, and G. Bosilca, "Reducing data motion and energy consumption of geospatial modeling applications using automated precision conversion," in *Proc. IEEE Int. Conf. Cluster Comput.*, 2023, pp. 330–342.



Qilong Pan received the bachelor's degree in statistics from Wuhan University of Technology, Wuhan, Hubei, China,, the master's degree in statistics from King Abdullah University of Science and Technology (KAUST), and the PhD degree in statistics from KAUST, in 2025, under the supervision of Prof. Ying Sun. His research interests include bridges statistical modeling, optimization, and high-performance computing (HPC) to tackle complex challenges in geospatial analytics and machine learning.



Sameh Abdulah (Senior Member, IEEE) received the MS and PhD degrees from The Ohio State University, Columbus, USA, in 2014 and 2016, respectively. He is currently a senior research scientist with the Applied Mathematics and Computational Science program, King Abdullah University of Science and Technology (KAUST), Saudi Arabia. His research interests include high-performance computing (HPC) applications, big data analytics, climate and weather modeling, large-scale spatial datasets, parallel spatial statistics, algorithm-based fault tolerance, and machine learning and data mining algorithms. He was a member of the KAUST team nominated for the ACM Gordon Bell Prize in 2022 and awarded the prize in 2024 (Climate Track) for their contributions to large-scale climate and weather modeling and prediction.



Mustafa Abduljabbar received the PhD degree in computer science from King Abdullah University of Science and Technology (KAUST). He was a research scientist with The Ohio State University, driving MVAICH-related projects and leading industry collaborations with Broadcom and ETRI (South Korea), and with Chalmers University of Technology, where he contributed to the EU's LEGATO initiative for energy-efficient heterogeneous computing. He is a senior member of Technical Staff with AMD, specializing in next-generation GPU communication and

networking libraries, with more than 15 years of experience in high-performance computing (HPC), parallel programming, and performance-oriented software development. His research—widely published and cited—spans distributed systems, energy-efficient computing, and performance modeling.



Hatem Ltaief received the master's in applied mathematics in 2004, and the PhD degree in computer science in 2007 from the University of Houston. He is currently the principal research scientist with the Extreme Computing Research Center, King Abdullah University of Science and Technology (KAUST), Saudi Arabia. His research interests include parallel numerical algorithms, parallel programming models, high-performance computing, and performance optimizations for multicore architectures and hardware accelerators. He was the recipient of several Best

Paper awards at the ACM PASC'18 and EuroPar'16 and '20 conferences, as well as the GAUSS award for Best Paper at ISC'20. He was among the Gordon Bell finalist teams at SC22, '23, and '24 and the awarded team for the Climate Track in 2024. He is the co-editor-in-chief for ACM Transactions on Mathematical Software and the IEEE TCPP coordinator for Africa, Europe, and the Middle East.



Andreas Herten is currently a researcher with Jülich Supercomputing Centre, heading the Accelerating Devices Lab within the Novel System Architecture Design Division, which he co-leads. He is part of the NVIDIA Application Lab at Jülich, in which he and his team analyze, port, and optimize scientific applications for GPUs. He is part of the core JUPITER team responsible for deploying the system. He is also involved in EU-funded projects for improving the performance of applications on GPUs and other accelerators.



Mathis Bode is currently a researcher for extreme-scale AI and simulation with the Jülich Supercomputing Centre. His activities included working with RWTH Aachen University, Stanford University, Caltech, University of Cambridge and his own startup (sold).



Matthew Pratola is currently a professor and chair with the Department of Statistics and a distinguished fellow, the Center for Applied Cybersecurity Research (CACR), Indiana University. His research interests include applied methodology, tree-based models, simulator-assisted inference, trustworthy statistical learning, and applications in physics, the environment, and health. His group works on methodology to combine subject-area simulators, such as physics simulators, with observational data to quantify uncertainties in simulator parameters and identify discrepancies between simulators and reality. His focus in recent years has been on investigating probabilistic frameworks for multiple assumed theories, i.e., multiple simulators. He has also worked extensively with Bayesian regression tree models, exploring various methodological developments and contributing open-source code to the community.



Arindam Fadikar received the PhD degree in statistics from Virginia Tech, USA, in 2019. He is currently an assistant computational statistician in the Decision and Infrastructure Sciences division, Argonne National Laboratory. He holds a scientist-at-large position with the University of Chicago, USA. His research interests include statistical inference of simulation models, calibration, and uncertainty quantification, with particular emphasis on epidemic applications.



Marc G. Genton received the PhD degree in statistics from the Swiss Federal Institute of Technology (EPFL), Lausanne, in 1996. He is currently an Al-Khawarizmi distinguished professor of statistics with the King Abdullah University of Science and Technology (KAUST), Saudi Arabia. His research interests include statistical analysis, flexible modeling, prediction, and uncertainty quantification of spatio-temporal data, with applications in environmental and climate science, as well as renewable energies. He is a fellow of the American Statistical Association (ASA), the Institute of Mathematical Statistics (IMS), and the American Association for the Advancement of Science (AAAS). He is an elected member of the International Statistical Institute (ISI). In 2010, he was the recipient of the El-Shaarawi Award for Excellence from the International Environmetrics Society (TIES) and the Distinguished Achievement Award from the Section on Statistics and the Environment (ENVR) of the American Statistical Association (ASA), ISI Service Award in 2019 and the Georges Matheron Lectureship Award in 2020 from the International Association for Mathematical Geosciences (IAMG). He led a Gordon Bell Prize finalist team with the ExaGeoStat software for Supercomputing 2022, the Royal Statistical Society (RSS) 2023 Barnett Award for his outstanding research in environmental statistics and the prestigious 2024 Don Owen Award from the ASA's San Antonio Chapter. He led a team that was awarded the 2024 ACM Gordon Bell Climate Modeling Prize for exascale climate emulators.



David E. Keyes (Senior Member, IEEE) received the BSE degree in aerospace and mechanical sciences from Princeton, in 1978 and the PhD degree in applied mathematics from Harvard in 1984. He directs the Extreme Computing Research Center, KAUST, where he was a founding Dean in 2009. He is currently a professor of applied mathematics, computer science, and mechanical engineering. He works at the interface between parallel computing, PDEs, and statistics. He led multi-institutional scalable solver software projects in the SciDAC and ASCI programs of the US Department of Energy (DoE), ran university collaboration programs at US DoE and NASA institutes, and taught at Columbia, Old Dominion, and Yale Universities. He is a fellow of SIAM, the AMS, and the AAAS. He has been awarded the Gordon Bell Prize from the ACM, the Sidney Fernbach Award from the IEEE Computer Society, and the SIAM Prize for Distinguished Service to the Profession.



Ying Sun received the PhD degree in statistics from Texas A&M University in 2011. She is currently a professor of statistics with King Abdullah University of Science and Technology (KAUST). At KAUST, she established and leads the Environmental Statistics Research Group, developing statistical models and methods for complex data to address important environmental problems. She has made original contributions to environmental statistics, particularly in spatiotemporal statistics, functional data analysis, visualization, and computational statistics, with an exceptionally broad array of applications.