Statistical Treatment of Variable MPI Latencies and MPI-Communication Hiding for Matrix-Free Finite Element Operators

Max Heldman
Virginia Tech
Department of Mathematics
Blacksburg, VA, USA
maxh@vt.edu

Johann Rudi
Virginia Tech
Department of Mathematics
Blacksburg, VA, USA
jrudi@vt.edu

Julie Bessac
National Renewable Energy Laboratory
Computational Science Center
Golden, Colorado, USA
Julie.Bessac@nrel.gov

Abstract

We consider large-scale implicit solvers for the numerical solution of partial differential equations (PDEs). The solvers require the high-bandwith networks of an HPC system for a fast time to solution. The increasing variability in performance of the HPC systems, most likely caused by variable communication latencies and network congestion, however, makes the execution time of solver algorithms unpredictable and hard to measure. In particular, the performance variability of the underlying system makes the reliable comparison of different algorithms and implementations difficult or impossible on HPC. We propose the use of statistical methods relying on hidden Markov models (HMM) to separate variable performance data into regimes corresponding to different levels of system latency. This allows us to, for example, identify and remove time periods when extremely high system latencies throttle application performance and distort performance measurements. We apply HMM to the careful analysis of implicit conjugate gradient solvers for finite-element discretized PDE, in particular comparing several new communication hiding methods for matrix-free operators of a PDE, which are critical for achieving peak performance in state-of-the-art PDE solvers. The HMM analysis allows us to overcome the strong performance variability in the HPC system. Our performance results for a model PDE problem discretized with 135 million degrees of freedom parallelized over 7168 cores of the Anvil supercomputer demonstrate that the communication hiding techniques can achieve up to a 10% speedup for the matrix-free matrix-vector product.

Keywords

Numerical Linear Algebra, Matrix-Free Finite Element Methods, Performance Variability, High-Performance Computing, Communication Hiding

ACM Reference Format:
Max Heldman, Johann Rudi, and Julie Bessac. 2025. Statistical Treatment of Variable MPI Latencies and MPI-Communication Hiding for Matrix-Free Finite Element Operators. In *2025 International Conference on Supercomputing (ICS '25), June 08–11, 2025, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3721145.3730413

1 Introduction

High Performance Computing (HPC) systems key capabilities are performant compute nodes combined with a performant network that exhibits low latencies and high bandwidths. Many applications in computational science and engineering rely on performant networks generally. Specifically in the context of the present work, HPC systems are well suited for the computational solution of large-scale implicit algebraic systems arising from discretized partial differential equations (PDEs) [26, 37, 41, 46]. Network latency and bandwidth are known to be critical for performance, because implicit PDE solvers exhibit lower computation to communication ratios relative to, for instance, dense matrix-matrix multiplication.

Compute nodes of HPC systems have been becoming increasingly capable as core counts and memory per node grow. In turn, this can increase pressure on network resources, and as a result the variability of network communication times has been observed to increase over the past decade [12, 27, 55]. Additionally, HPC systems are shared resources with hundreds or thousands of concurrently running jobs. This creates variability in the load on the network and possible communication contentions that occur sporadically and unpredictably. The increase in *variance* of communication times can diminish the performance of many applications to

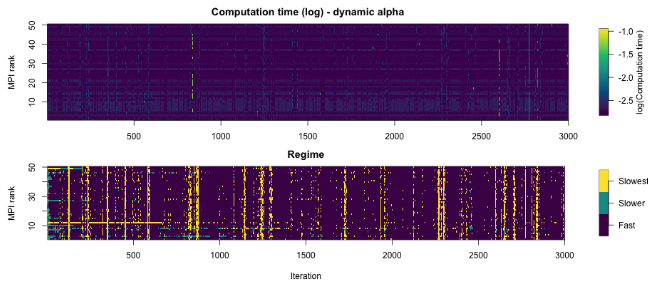

Figure 1: Top: Performance variability of `MatVec` timings across iterations of an implicit solve. Warmer colors indicate slowdowns and can affect most or all processes at one iteration, but isolated slowdowns on only a few processes are observed as well. Bottom: Classification of three regimes employed on the timing sequences (shown in top plot).

a much larger extent than an increase in the *mean* of communication times [12, 56]. The run-to-run variability has caused concerns [32] about confidence in and reproducibility of the reporting of performance results. In addition to network variability, dynamic performance variability can also be caused by the hardware or software on a compute node through various mechanisms (e.g., noise of the operating system and throttling of processor cores [22, 50, 55]).

The first purpose of this work is to address performance variability from the perspective of the application of implicit PDE solvers. We propose statistical treatment of the variability in high-frequency timers that instrument the algorithms of our solvers. In postprocessing, we can detect and filter timing measurements within our algorithms (in this case, matrix-vector products) into distinct regimes. This allows us to analyze the performance at different, realistic latency loads, which yields improved metrics for comparing the parallel performance of different implementations. With these statistical tools, the second purpose of this work is to devise new communication-hiding strategies to deal with variable and imbalanced communication latencies in performance-critical matrix-vector products.

1.1 Challenges of performance variability

We illustrate the challenges how performance variability affects the timing of iterations during an implicit solve, in this example using a (preconditioned) conjugate gradient method. The Krylov solver mainly consists of matrix–vector products (performed matrix-free for optimal computational efficiency, see Section 2) and vector–vector inner products. The communication is point-to-point during the matrix–vector multiplication; one Allreduce is required for each inner product. Since the algorithm is static, the sequence of computations and communications is identical for all iterations. Therefore, the expected time per iteration is expected to be constant up to (small) amounts of noise.

The illustration in Figure 1 (top), however, shows a drastically different picture. Outliers in iteration times appear sporadically, with extreme variation in magnitude. At a given time, they can be present on any number of ranks (from a handful to almost all ranks) within one iteration, and some ranks can be affected by milder slowdowns for many consecutive iterations. The variability across iterations is most likely caused by the HPC system and other jobs competing for resources, because of the static nature of the executed application. This results in strong differences between measured execution times from run to run. Therefore, we are facing significant challenges when the objective is to measure the performance of an application.

An ensemble study of many repetitions of the same numerical experiment would likely—though it is not guaranteed—converge to a summary statistic (e.g., median or mean runtime). In the presence of strong and frequent performance variability, this requires very large amounts of computational resources and energy usage, which would increase along with variability. Moreover, it incurs a high cost on developers and slows down progress.

1.2 Model problem

We consider the numerical solution of an elliptic PDE for scalar-valued functions $u : \mathbb{R}^d \to \mathbb{R}$, $d \in \{1, 2, 3\}$,

$$-\nabla \cdot (\kappa \nabla u) = f \quad \text{in } \Omega \tag{1}$$

with appropriate boundary conditions on $\partial\Omega$. The given forcing function $f : \mathbb{R}^d \to \mathbb{R}$ is on the right-hand side. The coefficient of the PDE (1) is a positive, spatially varying scalar field $\kappa : \mathbb{R}^d \to \mathbb{R}_+$. The Poisson Equation (1) frequently needs to be solved in the context of computational fluid dynamics, for example, for Stokes systems [46, 48] and Navier–Stokes systems [23, 25], therefore they serve as a benchmark for performance of implicit solvers [26, 43].

We discretize the computational domain Ω using hexahedral meshes that can be adaptively refined, where adaptive mesh refinement is based on the parallel forest-of-octree library p4est [16]. The PDE solutions in Equation (1) are discretized with continuous nodal finite elements of polynomial orders $k \geq 1$ and the discretization is nonconforming where the mesh is adaptively refined. Upon discretization, we need to solve an algebraic system of equations

$$\mathbf{A}\mathbf{u} = \mathbf{f} \tag{2}$$

with a sparse, positive definite matrix $\mathbf{A} \in \mathbb{R}^{N \times N}$, where the entries in the vector $\mathbf{u} \in \mathbb{R}^N$ are coefficients of the finite element basis functions.

The number of degrees of freedom (DOF), N, is typically very large and (2) is very poorly conditioned. For an efficient numerical solution, we employ preconditioned Krylov methods with multigrid as the preconditioner. Krylov methods and our multigrid implementation [46] only require the application of $\mathbf{A}$ to vectors, thus avoiding the construction and storage of $\mathbf{A}$ [20]. The performance-critical subroutine of the implicit solver becomes the matrix-free routine for applying the discretized PDE operator. This is referred to as the `MatVec`. Latencies in point-to-point communication within the `MatVec` often comprise the main bottleneck in parallel numerical solutions of Equation (2) at large scales. To reduce the cost of latency, we minimize the number of required messages and hide the communication by overlapping it with computations that are independent of communication. One contribution of the present work is new techniques for overlapping communication with computation during the parallel application of `MatVecs`.

1.3 Current state

The impact of performance variability on a compute node is confined to the job allocated on that node. This variability may introduce synchronization related delays with other compute nodes of the job that are not affected by any variability, but it does not directly impact the performance of applications that do not communicate with the impacted nodes. Performance interference primarily caused by compute nodes have been discussed in [22, 33, 50, 59].

Shared networks pose further challenges in achieving consistent run-to-run performance on large-scale HPC systems when exclusive, system-wide allocations are not feasible. Network interference between (independent) jobs is unlike the performance variability of compute nodes, because it can be greatly impacted by system behavior external to the job of interest. Multiple independent job allocations can interfere with each other through their shared network. Networks are increasingly encountering performance variability as their available bandwidth with respect to network injection rates decreases. A focus shift from compute node performance variability to the network is also observed in the literature [12, 18, 27, 29, 30, 55, 60]. The growing disparity between network traffic and communication latency and bandwidth leads to increased chances of network contention and network performance variability. In turn, the performance of applications is greatly impacted by the variability [18, 21, 30, 56, 63].

A research direction related to performance variability of applications is the monitoring of entire HPC systems [2, 52] and data centers [31], from an operator perspective. The goals in that body of research are extraction of features (i.e., telemetry data) and employing these in models for anomaly *detection* and their *diagnosis* or classification. A number of approaches for anomaly detection and diagnosis have been proposed [5, 14, 15, 36, 42, 54, 55, 61] along with approaches for benchmarking performance variability with synthetically generated system noise [17, 56]. Detection and diagnosis typically requires processing large amounts of data collected from the HPC system with the aim to improve the system performance and user experience. A major challenge is that processing needs to be in real time. The methods employed to process the data can be time-series models with regression fitting [34, 40], statistical and machine learning methods for unsupervised learning [1, 13, 31, 44, 62], and supervised learning [4, 7, 39, 55]. While the cited literature addresses the challenges arising from performance variability, especially those caused by network congestion, from a system operation perspective, our work differs by taking the vantage point of the application. In particular, our work takes the perspective that we do not have access to data of the system.

In our application, we computationally solve finite-element discretized PDEs based on fast and efficient parallel matrix-free `MatVec` routines within the context of a Krylov solver. Inside the `MatVecs`, overlapping the necessary point-to-point communication of shared discretization nodes with computations on nonshared nodes is critical to attain performance at large scales. Nonblocking scatter–gather algorithms for communication are used in [24, 25, 53]. Scatter communicates shared entries of the `MatVec`'s input vector among neighboring processes, and gather communicates the contributions from neighboring processes to the `MatVec`'s output vector. It is more efficient to modify the communication to gather-then-scatter if the input vector is always known to contain consistent data on the shared entries [35]. In that case, the communication phase of gather–scatter can be carried out subsequently and can be overlapped with the entire computation on nonshared discretization nodes.

The gather–scatter approach, however, can only be employed if the entire solver stack (including possible external libraries) maintains the consistency of shared entries in input vectors. If the assumption on input vectors is not possible, two communication phases are required. The computations on nonshared discretization nodes are then divided into two phases, each of which is overlapped with one of the communication phases. Existing finite element codes with two computation phases chose the computation volume to be equal for both phases (i.e., splitting the communication-independent computations in half) [6, 9, 38, 46].

In Section 2.4, we will propose two new methods for overlapping point-to-point communication within matrix-free FEM operator apply routines. Our results in Section 4 specifically compare the performance of matrix-vector products within a (preconditioned) conjugate gradient routine. Our research should not be confused with so-called pipelined Krylov methods (e.g., [49]), which are designed to reduce the synchronization of global communication (e.g., Allreduce) in Krylov iterations. In future work, we will investigate the combination of our point-to-point communication-hiding methods with pipelined Krylov methods to further improve the performance of the Krylov solver.

2 Methods and Contributions

2.1 Summary of Contributions

Statistical treatment of performance variability. We propose to use unsupervised learning with hidden Markov models and Gaussian mixture models for highly variable measurements of performance timings, which have a temporal dependence (see Section 2.2).

Measurement and analysis of detailed timing sequences across parallel processes. We instrument subroutines in our MatVec implementations (see Section 2.3) to emit detailed timing sequences for individual parallel processes (i.e., MPI ranks). We use our statistical tools to analyze the sequences in postprocessing. We separate realistic computation loads with variable communication latencies into distinct regimes.

Improved communication hiding in MatVecs. We propose two new methods for overlapping point-to-point communication with finite-element computations in MatVecs, called *dynamic α* and *MPI test*, which improve resilience to performance variability (see Section 2.4). We compare the performance to an established communication-hiding strategy, called *fixed α*. We are able to analyze the improvements of the proposed methods more reliably with our proposed statistical tools (see Section 4).

2.2 Statistical methods for variable background latencies

This section describes the statistical methods we use to identify latencies in the timing data. The timing data of implicit solver iterations k, $0 \leq k < n_{\text{it}}$ (with n_{it} total iterations) and MatVec subroutines within an iteration present a natural stochastic variability, because HPC systems are prone to performance variability across interconnected processes p, $0 \leq p < P$ (with P total processes) as introduced in Section 1. The statistical models we propose cluster the timing data $T_{p,k}$ of each MatVec on process $0 \leq p < P$ at iteration $0 \leq k < n_{\text{it}}$ into subpopulations that are constitutive of an overall population (entire dataset) without being given any other information about the subpopulation identification.

In practice, the clustering methods we use are applicable when multiple modes (i.e., maxima) are observed in the probability distribution of the entire population. Each mode represents the distribution maximum of a subpopulation. Figure 2 shows the probability density estimates of the median MatVec times over all parallel processes in three numerical

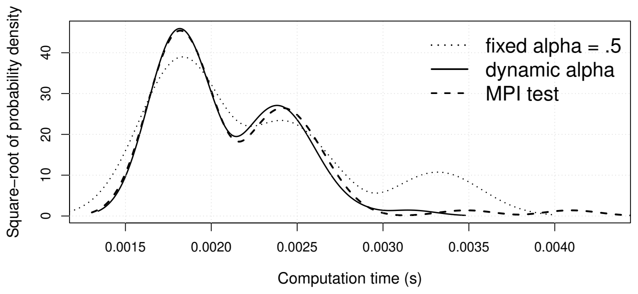

Figure 2: Probability density functions of median MatVec times, $\text{med}_k(T_{p,k})$, across processes p for the three methods: Fixed alpha (black), dynamic alpha (red) and MPI test (green). The square-root of probability density functions is displayed to enhance the visualization of small probability area.

experiments with several thousand iterations. Three modes can be observed in each experiment's distribution suggesting the use of mixture models. The three methods, which are introduced in Section 2.4, exhibit similar multimodality suggesting the use of mixture models with three subpopulations. Consequently, we propose the use of unsupervised statistical mixture models that represent multimodal data and characterize statistical properties of each subpopulation. As latencies can exhibit correlation across iterations, we will use Hidden Markov Models (HMM) [10, 11, 45] for which the hidden variable describing the belonging to a subpopulation is a Markov chain. In the following, we will refer to a subpopulation as regime, and we will denote the number of regimes by n_{reg}.

REMARK 1. *Our clustering methods also apply directly to the analysis of reduces time series $\hat{T}_k$ of $T_{p,k}$ computed across processes. In that case, we consider the time series $\hat{T}_k$ to fall into the special case $P = 1$. In our numerical experiments, we will consider the maximum reduction, $\hat{T}_k = T_k^{max} = \max_p(T_{p,k})$ taken across all processes for each iteration k.*

Mixture models represent the probability distribution of the entire population as a weighted sum of n_{reg} probability distributions $g(T) = \sum_{i=1}^{n_{\text{reg}}} \omega_i g_i(T)$, where g_i is represents the probability distribution of subpopulation/regime i, $1 \leq i \leq n_{\text{reg}}$. Often Gaussian distributions are used to represent each subpopulation g_i, or other parametric probability distributions can be used. In the following, we consider Gaussian distributions to model each subpopulation/regime $g_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ with mean μ_i and standard deviation σ_i. The weights ω_i for each distribution represent the unknown probability of occurrence of a regime. In order to match the mixture model parameters (weights ω_i, means μ_i, and standard deviations σ_i) to the data, these are estimated on the data during a calibration procedure, which is performed using maximum likelihood [19]. In the case of HMM, the transition matrix $\Pi \in \mathbb{R}^{n_{\text{reg}} \times n_{\text{reg}}}$ between regimes is also estimated during the calibration to data procedure. The resulting fitted mixture provides a description of each subpopulation's probability distribution and a probability for each datapoint to belong to each subpopulation. Typically, a datapoint is assigned to the regime with the highest probability. After the calibration (i.e., fit) of mixture models and HMM, the most likely regimes for each datapoint are estimated via the Viterbi algorithm [28, 58], which we employ hereafter. A proper calibration typically leads to regimes with distinct statistical properties pertaining to, but not limited to, their mean and variance.

The treatment of `MatVec` timing sequences with HMM faces the following three challenges. First, HMM has to be employed on data from P sequences that are recorded concurrently, where P is the (typically large) number of parallel processes. Second, a large P and a large count of iterations make it numerically challenging to accomplish fitting with maximum likelihood. Third, fitting needs to be robust to modifications to numerical algorithms and their implementations to allow a fair comparison between different computational techniques. We avoid fitting HMM on sequences of all P processes and provide a regime separation that is consistent across processes as well as across methods. This is achieved by subsampling a smaller number of representative processes and also subsampling the iterations used for fitting. Algorithm 1 summarizes our approach for fitting the `MatVec` timing sequences on parallel processes, and its practical details are discussed in Section 3.1. We note that HMM fitting for a reduced time series $\hat{T}_k$, as described in Remark 1, is simpler than considering $T_{p,k}$ with all processes. In particular, $\hat{T}_k$ does not require subsampling and the first two steps of Algorithm 1 are skipped.

Algorithm 1 Regime extraction with HMM & Viterbi

```
**Input:** MatVec timings $T_{p,k}$, $0 \le p < P$, $0 \le k < n_{\text{it}}$, of $P$ processes, $n_{\text{it}}$ iterations; number of regimes $n_{\text{reg}}$

$\mathbb{P}_{\text{sub}} \leftarrow$ select subset of processes representative for fitting
$\mathbb{I}_{\text{sub}} \leftarrow$ select iterations of MatVec timings
$\mu_i, \sigma_i, \Pi \leftarrow$ fit HMM($n_{\text{reg}}$) on data $T_{p,k}, p \in \mathbb{P}_{\text{sub}}, k \in \mathbb{I}_{\text{sub}}$
$\rho(T_{p,k}) \leftarrow$ estimate regimes via Viterbi using $\mu_i, \sigma_i, \Pi$
                                        ▷ for all $0 \le p < P$, $0 \le k < n_{\text{it}}$
```

When using HMM methods, one needs to select an appropriate number of regimes n_{reg} (subpopulations). The number of regimes is typically selected via quantitative indices such as Akaike information criterion (AIC) / Bayesian information criterion (BIC) [3] along with visual inspections. As discussed above, Figure 2 supports the choice of three regimes for the HMM to be fitted. Following this choice, the HMM is fitted on the subsampled dataset using the `R` package `hmmr` [57]. Outcome regimes of `MatVec` times are inspected in Section 3.1 to visually assess the realism (e.g., separate statistical properties in each regime) of the HMM fit.

2.3 Overview of the matrix-free FEM algorithm

Let $\Omega_p \subseteq \Omega$ be the subdomain associated with the computational process p. We partition the elements $\mathcal{E}$ on p into three groups, $\mathcal{E}_1$, $\mathcal{E}_2$, and $\mathcal{M}$, $\mathcal{E} = \mathcal{E}_1 \cup \mathcal{E}_2 \cup \mathcal{M}$. The first two groups, $\mathcal{E}_1$ and $\mathcal{E}_2$, are *local elements* that do not touch the interprocess boundaries, and $\mathcal{M}$ are *mirror elements* that do touch the interprocess boundaries. During the application of a finite element operator $\mathbf{A}$ to a vector $\mathbf{u}$, the computation on $\mathcal{E}_1$ is overlapped with the input communication of degrees-of-freedom on elements in $\mathcal{M}$ that are *not* owned by p. Similarly, the computation on $\mathcal{E}_2$ is overlapped with the output communication, which are contributions from neighboring processes to elements on $\mathcal{M}$ owned by p. The general algorithm is presented in Algorithm 2 below.

Algorithm 2 Parallel `MatVec`

```
Start **MPI receive**: non-owned DOFs of $\mathbf{u}$ on $E \in \mathcal{M}$
**MPI send** contributions to owned DOFs on $\partial\Omega_P$
Start **MPI receive**: owned DOFs on $\partial\Omega_P$
**for** $E \in \mathcal{E}_1$ (non-mirrors, 1st block) **do**
    Compute contributions to $\mathbf{Au} = \mathbf{b}$ on $E$
**end for**
Finish **MPI receive**: non-owned DOFs of $\mathbf{u}$ on $E \in \mathcal{M}$
**for** $E \in \mathcal{M}$ (mirror elements) **do**
    Compute contributions to $\mathbf{Au} = \mathbf{b}$ on $E$
**end for**
Start **MPI send**: non-owned DOFs of $\mathbf{u}$ on $E \in \mathcal{M}$
**for** $E \in \mathcal{E}_2$ (non-mirrors, 2nd block) **do**
    Compute contributions to $\mathbf{Au} = \mathbf{b}$ on $E$
**end for**
Finish **MPI receive**: owned DOFs of $\mathbf{b}$ on $\partial\Omega_P$
```

The elements in $\mathcal{M}$ and $\mathcal{E}_1 \cup \mathcal{E}_2$ are determined by the parallel partition, which for the purpose of this work we consider to be a black box. However, the subgroups $\mathcal{E}_1$ and $\mathcal{E}_2$ can be freely chosen for the implementation. In many finite element implementations (e.g., in [38]), the elements in $\mathcal{E}_1$ and $\mathcal{E}_2$ are chosen to have equal size, thus $|\mathcal{E}_1| \approx |\mathcal{E}_2|$. This is a reasonable choice if one does not have any information about the expected input and output latencies. Indeed, except in extreme cases (e.g., process p has no non-owned DOFs) the balance of time spent waiting for input and output on a particular process can be difficult to determine; it depends on not only the relative number of owned and non-owned

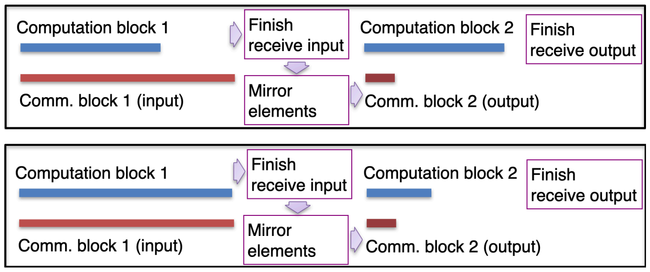

Figure 3: Top: `MatVec` algorithm with $\alpha = \frac{1}{2}$. The blue computation blocks are divided evenly and only partially overlap input and output communication. Bottom: `MatVec` algorithm with dynamically adjusted α. The blue computation blocks are divided to exactly overlap input and output communication.

DOFs on the process boundary, but also on the number of communication partners and the parallel configuration.

2.4 Communication hiding methods

In this section, we consider several strategies for optimizing the splitting $\mathcal{E}_1 \cup \mathcal{E}_2$ on each parallel process, other than the baseline strategy of $|\mathcal{E}_1| \approx |\mathcal{E}_2|$, to optimize the overlap of communication and computation during the `MatVec`. We encode the splitting using a parameter α, $0 \leq \alpha \leq 1$, such that $|\mathcal{E}_1| = \lfloor \alpha |\mathcal{E}_1 \cup \mathcal{E}_2| \rfloor$ and $|\mathcal{E}_2| = \lceil (1-\alpha)|\mathcal{E}_1 \cup \mathcal{E}_2| \rceil$. We emphasize that we aim to find an optimal splitting for every process p, so that each process p will use a different value of α. The baseline strategy described in Section 2.3 sets $\alpha = \frac{1}{2}$ for all processes; this strategy is depicted in Figure 3 (top). In the illustrated case, because the time to receive input communication is significantly longer than it is for the output, the even splitting is not optimal and the solver needs to wait to receive input (during the portion of communication block 1 extending beyond computation block 1). The bottom image rectifies this problem by choosing $\alpha > \frac{1}{2}$, so that a greater portion of computation time is spent in the first half. The splitting in Figure 3 (bottom) is optimal, since both communication blocks are now hidden by computations. In addition, output communication can only be sent after mirror element computations are completed. Hence, although one could further extend the first computation block and still completely overlap the first communication block, doing so would delay the whole `MatVec` by delaying the initiation of output communication.

In practice, the optimal value of α on a given process can be influenced by many factors, including the topology of the computing environment and the local mesh geometry, making it difficult to determine the balance of input and output communication *a priori*. In addition, altering the value of α on one process can affect the optimal value of α on other nearby processes, as observed communication latencies are not independent. Therefore, we choose to take an empirical approach, using a fixed-point routine for updating α at runtime based on observed input and output waiting times, T_k^{in} and T_k^{out} at iteration k. In an MPI implementation, the waiting times measure the duration of the call to the `MPI_Waitall` function. The new algorithm has three parameters: the initial value for α, α_0, which we always take to be $\frac{1}{2}$, $\theta \in [0,1]$, which determines the size of each increment or decrement to α, and a zero-latency threshold $0 < \varepsilon \ll 1$. The algorithm is presented in Algorithm 3. We refer to the new method as the dynamic α technique, because α is updated throughout a computation based on an observed pattern of latencies.

Algorithm 3 Update of splitting parameter

```
$r^{k-1} = \frac{\alpha^{k-1}}{1-\alpha^{k-1}}$
**if** $T_k^{\text{in}} > \varepsilon$ and $T_k^{\text{out}} < \varepsilon$ **then**
    $r^k = r^{k-1} \frac{T_k^{\text{in}}}{\varepsilon}$
**else if** $T_k^{\text{in}} < \varepsilon$ and $T_k^{\text{out}} > \varepsilon$ **then**
    $\alpha^k = \frac{r^k}{1+r^k}$
**end if**
$\alpha^k = \theta\alpha^k + (1-\theta)\alpha^{k-1}$
```

The idea of the algorithm is that, when $T_k^{\text{in}} > \varepsilon$ is large and $T_k^{\text{out}} < \varepsilon$, then the latency of the output communication block is already entirely hidden but the latency of the input communication is not. Therefore, in that case we can improve the overlap by shifting some elements from $\mathcal{E}_2$ to $\mathcal{E}_1$. This is the scenario depicted in Figure 3 (bottom). Conversely, when $T_k^{\text{in}} < \varepsilon$ is small and $T_k^{\text{out}} > \varepsilon$ is large, then we can improve the overlap by shifting some elements from $\mathcal{E}_1$ to $\mathcal{E}_2$. We shift the splitting variable indirectly by adjusting $r^k = \frac{\alpha^{k-1}}{1-\alpha^{k-1}} = \frac{|\mathcal{E}_1|}{|\mathcal{E}_1|+|\mathcal{E}_2|}$, which is the percentage of the total number of local elements in $\mathcal{E}_1$. Then, we apply damping using the factor θ to avoid too rapid oscillations of α, because our goal is to improve the `MatVec` performance on the *modal* communication pattern.

When either of the conditions

$$T_k^{\text{in}} > \varepsilon \quad \text{and} \quad T_k^{\text{out}} < \varepsilon$$
$$T_k^{\text{in}} < \varepsilon \quad \text{and} \quad T_k^{\text{out}} > \varepsilon$$

is satisfied, we refer to T_k^{in} or T_k^{out} as excess waiting time. The zero latency threshold, ε, is chosen empirically so that it is a fraction of the overall computation time available to cover the communication. The size of the shift depends on the magnitude of the input and output communication; if the communication times are larger (smaller) then the adjustment becomes larger (smaller). In other cases, when there is no excess waiting time, we do not change α because the communication is already completely overlapped with computation or the communication time is too large to be hidden.

In Section 4, we compare the performance of `MatVec` operations computed using the dynamic α method and the baseline fixed α method. Note that the fixed α method is the current state of the art, as far as the authors are aware (see the discussion at the end of Section 1.3). We also introduce a third method, MPI test, which uses the function `MPI_Test_all` to check for the completion of the input communication. When the input communication is complete, the computation on the mirror elements is immediately started.

In Figure 5, we show the evolution of the splitting parameter (bottom image) for the three different methods over the course of 8,192 `MatVec` operations during a single run of the CG algorithm, presented in Section 4. The splitting parameter we record for the MPI test method is the location where the first element loop was terminated. The value of α is therefore *inferred* after each process-local `MatVec`, rather than specified before the `MatVec`. We emphasize here the main distinction between the two new methods: the dynamic α method uses a predetermined splitting for each `MatVec`, and makes a damped update to the splitting for the next `MatVec` based on observed input and output waiting times. In contrast, the MPI Test method does not use a predetermined splitting at all, instead adapting instantly to changing communication latencies. In practice, this means that the dynamic α method adapts to *modal* communication patterns. Our numerical results (e.g., Figure 11 and Figure 12) suggest that the two methods perform similarly; this is because most `MatVec` iterations follow the modal communication pattern.

The top two rows in Figure 5 show the waiting times for input (top row) and output (bottom row) communication during the same run, with the `MatVec` operations with excess input or output waiting times colored in blue. The dashed black line represents the zero-latency threshold $\varepsilon = 10^{-4}$, which is roughly $\frac{1}{10}$ the approximate non-mirror element computation time of 10^{-3}. The dashed red line indicates the mean value of the waiting time over the course of the run. For all three runs, the input communication experiences almost exclusively latencies below the zero-latency threshold ε, and the output latencies are frequently slightly above ε, which leads to excess output waiting. For the dynamic α method, this means that α adjusts to be close to zero, while the flexibly adjusting MPI Test method also almost always splits toward zero. The mean output waiting time for the dynamic α method is reduced closer to the zero-latency threshold compared with the fixed α method. We can see the effect of adjusting α in time, as the first the few `MatVec` operations, when α is close to $\frac{1}{2}$ have a larger waiting time than the later iterations.

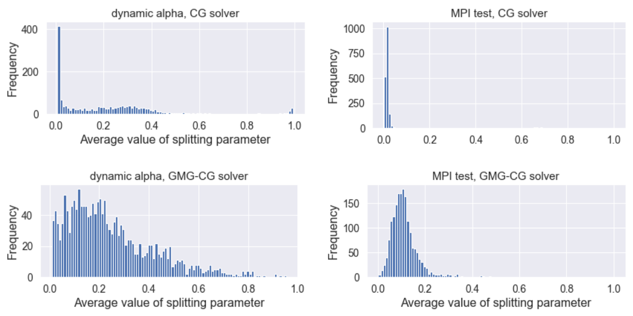

Figure 4: Mean splitting parameter for the dynamic α method (left) and MPI test method (right) over the course of a CG run (top) and GMG-CG run (bottom).

In Figure 4 below, we show the final distribution of the splitting parameter on the first 1792 of the 7168 MPI ranks used in the CG solve (top) and a separate GMG-CG solve, i.e., a CG solve using geometric multigrid as a preconditioner (bottom). The distribution of the GMG-CG splitting parameter is only shown for the finest grid of the multigrid hierarchy.

We see that for the CG solve, α is heavily concentrated near zero for both the dynamic α and MPI test methods. For the GMG-CG runs the distribution concentrated at a slightly higher value. This likely reflects the fact that for GMG-CG, the balance of input and output communication is not as consistent because the processes can become unsynchronized after the first smoothing step. In particular, processes which finish their coarse grid computations earlier need to wait longer for input communication.

3 Experimental Setup and Hardware Platforms

We present the setup of numerical experiments, and the hardware and software environments. The goal is to ensure the reproducibility of the work and to introduce the computational studies described in the results Section 4.

3.1 Statistical regime extraction for variable performance

Algorithm 1 is implemented on the data with the following specifications. For per-process data $T_{p,k}$, we pick a subset of processes $\mathbb{P}_{\text{sub}}$ for fitting by equidistantly selecting 20 processes across all P processes sorted by their median `MatVec` times (taken over iterations). 20 samples are sufficient to capture the variety of timing statistics, because we observe three major modes in the total population of `MatVec` timings.

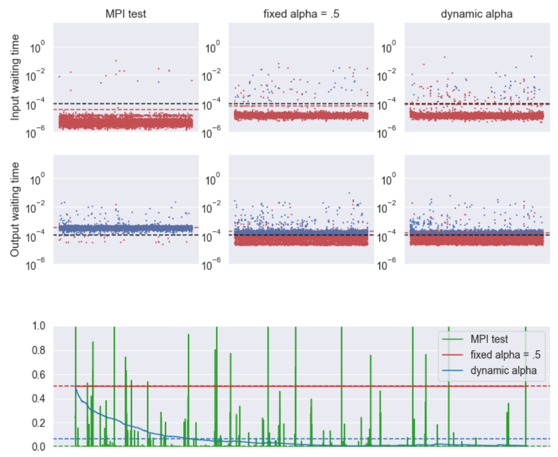

Figure 5: Top: Waiting times for input (top row) and output (bottom row) communication during a single run of the CG algorithm. The dashed black line represents the zero-latency threshold ε, while the dashed red line indicates the mean value of the waiting time over the course of the run. The `MatVec` operations with excess input or output communication are colored blue. Bottom: Evolution of the splitting parameter α over the course of 8,192 `MatVec` operations during the same run.

We furthermore subsample iterations, denoted by the set $\mathbb{I}_{\text{sub}}$, for each process in $\mathbb{P}_{\text{sub}}$. The choice of subsampling is corroborated by Figure 1 (top), which displays the timing sequences across all iterations (x-axis) and across 100 processes (i.e., MPI ranks, on y-axis). In the following, HMM is fitted to each subsampled datasets of process-local `MatVec` times of size $|\mathbb{P}_{\text{sub}}| \times |\mathbb{I}_{\text{sub}}| = 20 \times 8192$, for each numerical method (see Section 2.4). However, when studying the maximum `MatVec` time $T_k^{\max} = \max_p(T_{p,k})$ across all processes (see Remark 1), we repeat the same steps described above, but skip the process selection step and we fit HMM to the full time-series data $\{T_k^{\max}\}_{k=1}^{8192}$. The timing results are fitted independently for different algorithms, because their computation throughput and communication latencies are expected to differ.

To assess the realism of the HMM fit on the timing sequences, we visualize the associated regimes estimated from Algorithm 1 and investigate statistics in each regime. We expect regimes to exhibit different properties of latency, especially regimes gathering faster and slower `MatVec` timings. Figure 6 displays HMM fitted `MatVec` times for the three methods from Section 2.4 applied to CG and GMG-CG experiments (see Section 3.2). Each HMM validation plot shows a clear separation between the three regimes. The regime in

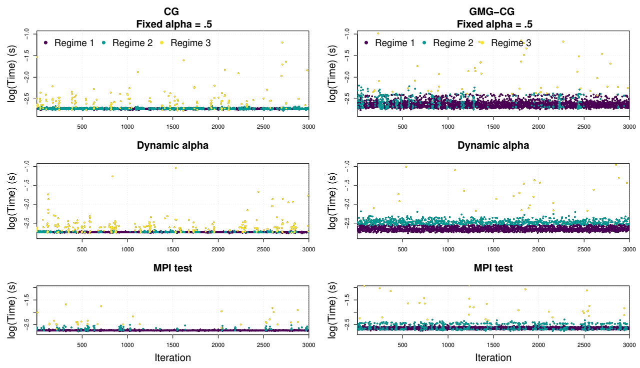

Figure 6: `MatVec` times across iterations for the median MPI rank across the 20 training MPI ranks for the three communication hiding methods: fixed α (top), dynamic α (center), MPI test (bottom). Left panels show results for CG and right ones show results for GMG-CG. Regimes are indicated by color: fast (dark purple), slower (cyan) and slowest (yellow).

Table 1: Regime statistics (mean μ_i and standard deviation σ_i) of `MatVec` times in each regime ($i = 1, 2, 3$) and proportion of iterations/`MatVecs` (% it.) spent in each regime for the three studied methods applied to CG.

	Fixed alpha			Dynamic alpha			MPI test		
i	μ_i	σ_i	% it.	μ_i	σ_i	% it.	μ_i	σ_i	% it.
1	1.79e-03	5.89e-05	0.33	1.75e-03	5.12e-05	0.34	1.78e-03	4.30e-05	0.35
2	1.89e-03	5.49e-05	0.32	1.87e-03	5.22e-05	0.31	1.90e-03	5.61e-05	0.32
3	2.89e-03	6.19e-03	0.35	2.76e-03	6.00e-03	0.35	2.72e-03	4.28e-03	0.33

dark purple gathers faster times (dark purple dots) with a smaller dispersion compared to the slower times (cyan dots); the slowest times (yellow dots) have more dispersion and successfully capture the higher variability in the timings. This clear separation of regimes indicates a reliable fitting of the HMM on the data. Additionally, Table 1 displays statistics (mean and standard deviation) of the `MatVec` timing and proportion of iterations spent in each regime for the three studied methods applied to CG experiments. Complementing Figure 6, Regime 1 is the fastest with typically least dispersion whereas Regime 3 is the slowest with largest dispersion. The slowest regime is dominated by slow times that arise because of performance variability of the HPC system; we believe it is likely caused by high peaks of communication latency or network congestion. Consistent results are also observed for GMG-CG (not shown). Finally, Figure 6 and Table 1 provide consistent results in terms of physical meaning of the regimes across the three studied methods (fixed α, dynamic α and MPI test) and across the algorithms they are applied to (CG and GMG-CG). This highlights the robustness of Algorithm 1 with respect to reliable extraction of regimes across different communication methods and for the algorithms that we will analyze in Section 4.

3.2 Numerical experiments for implicit PDE solvers

To measure the performance of our communication hiding algorithm, we apply the conjugate gradient (CG) solver and geometric multigrid (left)-preconditioned CG solver (GMG-CG) to the model Poisson equation (1). For the spatial domain, we use the unit cube $\Omega = [0,1]^3$ subdivided into a uniform, hexahedral mesh with 2^{27} =134,217,728 elements with 135,005,697 unknowns in our piecewise trilinear finite element discretization. We control the number of CG iterations by choosing a diffusion coefficient κ with a jump discontinuity of several orders of magnitude, which increases the condition number of the stiffness matrix, preventing the solver from converging before the maximum iteration count is reached. We note that the choice of κ does not influence our performance results.

Our parallel partition on the Anvil HPC system (see details in Section 3.3) consists of 56 nodes with 128 MPI ranks per node, for a total of 7168 MPI ranks with approximately 18,720 elements and 18,830 unknowns per rank. We chose the node count to be the maximum available on Anvil, in order to have the most realistic, and most challenging, parallel environment possible for our experiments. On the Frontera HPC system, we used a partition of 64 nodes with 56 MPI ranks per node, resulting in 3,584 total MPI ranks. On both systems, we chose the problem size per core to approach, but not reach, the strong scaling limit of our application so that we would have an amount of computation commensurate to the network latency — if the ratio of available computation relative to the latency is too high, for example, then the $\alpha = \frac{1}{2}$ method will be able to hide all of the communication and our new techniques will not be needed. In future work, we plan to investigate the effect of our communication hiding methods on a wider range of problems by varying per-core problem sizes, node counts, and network architectures.

Our software stack includes the PETSc library [8] for the CG solver and the Chebyshev-accelerated Jacobi multigrid smoothers. We use our own implementation of `MatVec` apply routines for the stiffness matrix and the matrix-free operators for interpolation and restriction within multigrid. For the coarse grid solve we use one V-cycle of geometric multigrid with PETSc's GAMG (algebraic multigrid) as the solver at the coarsest geometric grid. As previously mentioned, the parallel mesh generation is handled by the forest-of-octrees library p4est [16].

During 10 runs of each solver, we record the waiting time for input and output communication during the first 8192 matrix-vector products computed on the first 1792 MPI ranks (totally $5.8 \cdot 10^7$ process-local `MatVec` operations recorded per run), along with the time spent computing over the first and second groups of non-mirror elements. Referring to Algorithm 2, this means we record the time $T^{\text{in}}_{p,k}$ as the total time elapsed between the start of the computations over the first block of non-mirror elements and the end of the first MPI receive call, and the time $T^{\text{out}}_{p,k}$ as the time elapsed between the start of the start of the second block of non-mirror computations. Our timing data do not include mirror element computations, since these do not depend in any way on the splitting, and we refer to $T_{p,k} = T^{\text{in}}_{p,k} + T^{\text{out}}_{p,k}$ as the `MatVec` time. For the smoothing steps, we only record the `MatVec` timings on the finest mesh, and we report all of the GMG-CG `MatVec` data as an aggregate instead of separating the timings for the pre- and post-smoothing steps and the energy inner product needed for the Krylov iteration.

Table 2: Specifications of RCAC's Anvil HPC system.

Processors	AMD EPYC 7763 64C 2.45GHz "Milan"
Interconnect	Mellanox InfiniBand HDR100
CPUs per node	2
CPU cores per node	128
Hardware threads per core	1
CPU clock rate (nominal)	2.45 GHz
Memory per node	256 GB
C/C++ compiler	GCC 11.2.0
MPI library	OpenMPI 4.0.6
BLAS library	OpenBLAS 0.3.17

Table 3: Specifications of TACC's Frontera HPC system.

Processors	Intel Xeon Platinum 8280 28C 2.7GHz "Cascade Lake"
Interconnect	Mellanox InfiniBand HDR100
CPUs per node	2
CPU cores per node	56
Hardware threads per core	1
CPU clock rate (nominal)	2.7 GHz
Memory per node	192 GB
C/C++ compiler	Intel 19.1.1
MPI library	Intel MPI 19.0.9
BLAS library	Intel MKL 2020.1

3.3 HPC systems

Computations are carried out on two systems: the Anvil HPC system at the Rosen Center for Advanced Computing (RCAC) and the Frontera HPC system at the Texas Advanced Computing Center (TACC). Table 2 lists the hardware and software specifications of Anvil and Frontera, repsectively. The software environment consists of C/C++ compilers, an MPI library, and a BLAS/LAPACK library, as well as the libraries PETSc [8] and p4est [16].

4 Performance Results

Performance of the communication hiding methods is presented with three granularities of measurements. First, we look at the distribution of computation times on the Anvil supercomputer sampled from individual MPI ranks and taken as a whole (see Section 4.1). Second, we summarize the distribution of these computation times on each MPI rank, still viewing each rank individually. Finally, we aggregate the computation times across MPI ranks by measuring the maximum time per MatVec per rank (see Section 4.3). The third measure (i.e., maximum across ranks) is influenced by the first two, but is the most important measure, especially when each MatVec alternates with (blocking) collective communication, such as the inner product in the CG algorithm. In that case, it gives a good estimate for the expected bottom-line speedup for the solver without the additional noise from other CG operations, such as the collective Allreduce call.

In Section 4.4, we demonstrate our clustering method when applied to the maximum computation time across MPI ranks, this time using MatVec timing data from the Frontera supercomputer. The Frontera timings feature an extreme, temporally extended slowdown event which would make it infeasible to compare the raw times of the different algorithms without clustering. This contrasts with the Anvil results, where the slow MatVecs typically appear intermittently. From the fast regime data, we extract average per MatVec timings that show clearly the overall improvement of the new algorithms on Frontera.

4.1 Fine granularity performance: time per MatVec per MPI rank

The purpose of this section is to compare individual, process-local MatVec times between the three methods. The results give insight not only into the relative performance of the methods, but also into some of the details for how they work to improve MatVec execution times. The scatter plots in the top row and histograms in the bottom row of Figure 7 reflect a sample of 10 process-local MatVec times on each MPI rank for the fastest run of the CG (left panel) and GMG-CG (right panel) solves. Each data point represents one MatVec on one rank. The histograms contain the sums of the input and output computation times of the points in the upper plots (for the exact meaning of those times, see Section 3.2).

Looking at the top two plots of Figure 7, we observe that the fixed α method has a distribution concentrated around 10^{-3} seconds for both input and output time. These points reflect zero or near-zero latency MatVec times, where the waiting time is significantly shorter than the computation time and the latency is effectively hidden. Likewise, the sharp line at $\sim 10^{-3}$ bounding the input and output computation times from below in the fixed α distribution reflects the time to compute over half of the non-mirror elements — this is the *minimum* possible time for $T^{\text{in}}_{p,k}$ or $T^{\text{out}}_{p,k}$ for the fixed α method, achieved when there is no input or output latency. On the other hand, the MPI test and dynamic α methods, which adjust the amount of computation based on the latency and expected latency, respectively, have a lower minimum amount of computation time for input and output depending on how many elements are shifted from one computation

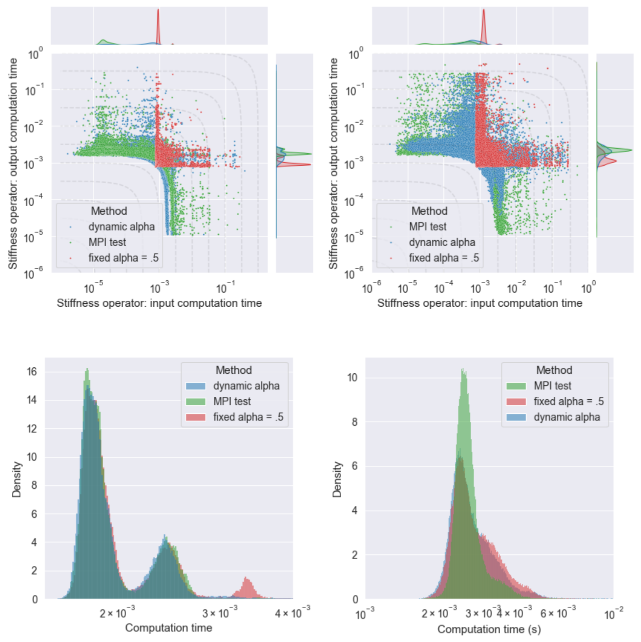

Figure 7: Scatter plot (top row) and histogram (bottom row) of randomly sampled process-local MatVec times for the fastest run of the CG (left column) and GMG-CG (right column) solves. The scatter plots are partially surrounded by two marginal density plots along the horizontal and vertical axis, respectively.

block to the other. We also note that the MPI test method has a slightly larger amount of input computation time for MatVec operations with small output computation times, which reflects the overhead of checking for completion of the input communication during the first block of non-mirror element computations.

Moving to the histograms in the bottom row of Figure 7, for the CG experiment we can see three clearly separated regimes: the three modes, interpreted from left to right, reflect MatVec operations where the latency is almost completely hidden, partially hidden, and hardly hidden at all (typically extremely high latencies). In contrast, the GMG-CG times appear as a single mode. We expect that the unimodal distributions for GMG-CG result from the absence of global synchronization during each multigrid V-cycle, which makes the behavior less predictable and less likely to separate into distinct modes of regimes. We note that for both the CG and GMG-CG histograms, the high latency MatVec times extend much further to beyond the horizontal axis limits, which we have constrained for readability.

Comparing the scatterplots and the scale of the histogram axes for CG and GMG-CG, we see that MatVec times several times higher than the minimum are much more typical for

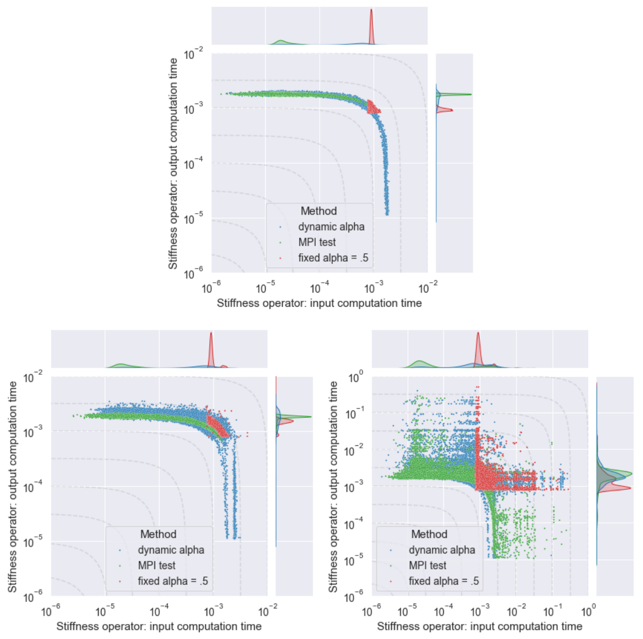

Figure 8: Scatter plots of the `MatVec` times of the CG iterations from Figure 7 separated into three regimes, as described in 2.2: the fastest regime (top), a slower regime (bottom left), and the slowest regime (bottom right).

the GMG-CG solver. We can see that the computation times for the CG experiment are shifted slightly to the left for the dynamic α and MPI test methods compared with the fixed α method. The shift is more noticeable for the GMG-CG experiment, where the MPI test results in particular are significantly more concentrated than the fixed α results.

The analysis of Figure 7 can be enhanced by considering the regimes as described in Section 3.1. We show the scatter plots of the regime data for the `MatVec` times of the CG iteration in Figure 8. The `MatVec` times clearly divide into three categories: first, the fastest regime (top) has computation times with essentially completely hidden latency; second, the slower regime (bottom left) contains mainly times with moderate latencies; and, third, the slowest regime (bottom right) contains times with extremely high latencies. The influence of performance variability from network latencies is clearly extracted in the slowest regime, while the first two regimes have few large outliers. Compared to the unseparated data shown in Figure 7, the first two regimes provide better insights into the different behaviour the methods in practice, which we would be challenging to decipher otherwise.

We observe that, toward the end points of the dynamic α and MPI test regime 1 and regime 2 distributions, the two

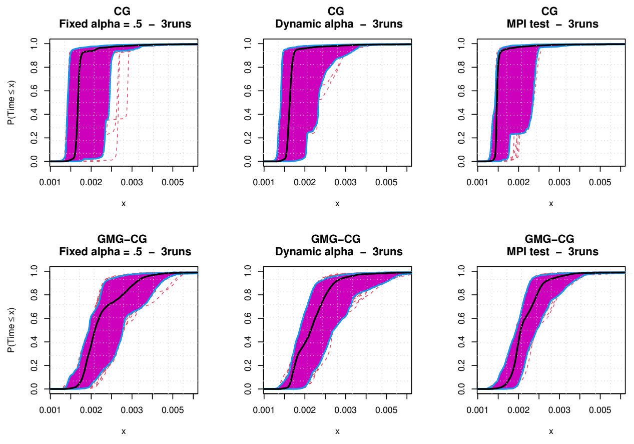

Figure 9: Functional boxplots of cumulative distribution functions of `MatVec` times across MPI ranks. Cumulative functions are computed for each communication method (from left to right: fixed α, dynamic α and, MPI test) applied to CG (top) and GMG-CG (bottom) solvers. Black curve is the median CDF curve, purple region contains 50% of the CDF curves and dashed lines are outlier CDF curves.

methods are both able to decrease the input (output) computation time without increasing the output (input) computation time. We also note that the dynamic α and MPI test also spend a greater proportion of time in the fast regime—71% and 75%, respectively, compared with 47% for the fixed α method—because they mask more of the latency.

4.2 Intermediate granularity performance: `MatVec` distribution per MPI rank

In this section, as a bridge between the previous section and Section 4.3, we summarize the improvement per MPI rank in Figure 9. We will show that the dynamic α and MPI test methods have a more significant improvement on the overall runtime than the distribution of process-local `MatVec` times from Section 4.1 would suggest. The result is due to the fact that viewing process-local `MatVec` times *individually* (i.e., treating each one as an equally important data point) does not account for cases where one slow MPI rank slows down the computation on all other MPI ranks. The functional box plots in Figure 9 capture the essence of the per-rank improvement via the cumulative distribution functions (CDF) of `MatVec` times. The black curve on each plot represents the median CDF, while the purple region represents the span of the CDFs between the 25th and 75th percentiles. Functional boxplots rely on the band-depth metric that measures the centrality of a curve with respect of an ensemble of curves [51]. Via this metric, one can order curves by their centrality

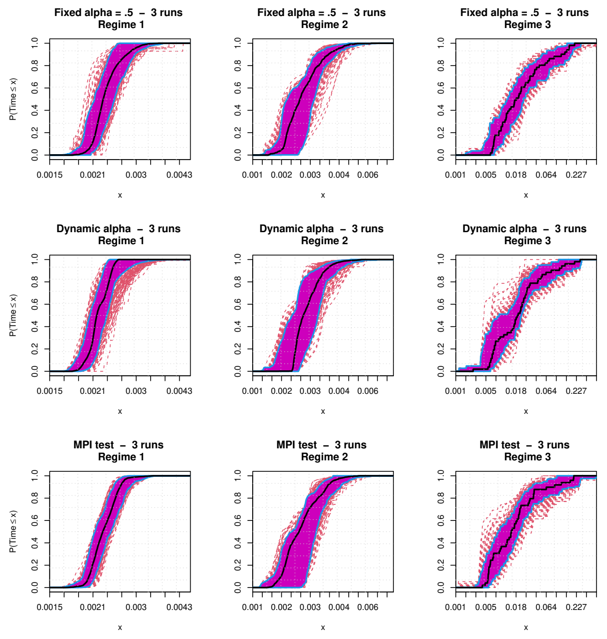

Figure 10: Functional boxplots as in Figure 9 for MatVec times CDFs computed in each regime extracted as in Section 3.1 for the GMG-CG solver. Regimes 1 to 3 are shown from left to right and the three communication-hiding methods (fixed α, dynamic α and MPI test) are shown from top to bottom.

and create concepts similar to quantiles and their associated visualization, namely boxplot. When visualizing the displayed CDFs, one expects the curves to reach the value 1 on the y-axis for the fastest times (most left values on x-axis) indicating more probability mass (across MPI ranks) allocated to faster `MatVec` times. For both CG and GMG-CG, the dynamic α and MPI test methods have a median CDF that is shifted to the top left corner compared with the fixed α method, which represents overall faster `MatVec` times for dynamic α and MPI test. Additionally, purple regions show less spread for dynamic α and MPI test methods than for the fixed α method, indicating more consistency across MPI ranks for these methods. Finally, the outermost CDFs in the envelope (dotted lines in each image) are significantly shifted to the left for the dynamic α and MPI test methods. This supports that the two new methods have the most significant impact on the overall slowest MPI ranks, which ultimately has the greatest impact on the solver performance.

Figure 10 shows a refinement of Figure 9 (bottom row) for GMG-CG experiments, where CDFs are computed in each regime extracted as in Section 3.1. Similarly, we observe CDF curves shifted to the top left corner for the dynamic α and MPI test methods in each regime, especially the fastest regimes 1 and 2. Regimes 1 and 2 also exhibit a lesser spread in the CDF envelope indicating more consistent computation times across ranks. Consistent results are also seen on the CG algorithm (not shown here).

Figures 9 and 10 show statistics from three runs. The statistics do not exhibit inconsistency (e.g., clustering of CDF from each run) between the runs, suggesting that `MatVec` times have similar distributions across runs. To corroborate this, for each of the three communication techniques (fixed α, dynamic α, and MPI test) and solver types (CG and GMG-CG), we applied a Kolmogorov-Smirnov test to compare the probability distributions of `MatVec` times between different runs. Results indicate that different runs using a particular solver and communication method can be considered as samples from the same distribution, whereas runs using different communication methods do not. We note that infrequent and significant changes in communication patterns (e.g., the major network slowdown observed in the Frontera results in Section 4.4) could cause the distributions of `MatVec` times to differ between runs. However, we expect that after removing the outliers (by, e.g., separating the data into regimes), the remaining data will still be drawn from the same distribution.

4.3 Coarse granularity performance: maximum time per MPI rank

The left image in Figure 11 shows the cumulative sums of the maximum time per `MatVec`, taken over all MPI ranks at one iteration, for the CG solver. We combine the fastest three runs for each communication-hiding method. The plotted quantity is

$$S_K = \sum_{k=1}^{K} T^{\max}_{\sigma(k)},$$

where $\{T^{\max}_{\sigma(k)}\}_{k=1}^{n_{\text{it}}}$ is a smallest-to-largest sorted version of $T^{\max}_k$, itself defined by reducing $T_{p,k}$ over the processes p:

$$\{T_k\}_{k=0}^{n_{\text{it}-1}} \quad \text{with} \quad T_k = \max\{T_{p,k}\}_{p=0}^{P-1}.$$

The cumulative sums are similar to a CDF plot, in that they reveal the difference in the performance of each method for fast and slow `MatVec` times.

Labeling the cumulative sum value for fixed α, dynamic α, and MPI test by $S_K^{1/2}$, S_K^{α}, and S_K^{test}, respectively, the dashed lines in Figure 11 (left) show the percentage differences $\frac{S_K^{1/2}-S_K^{1/2}}{|S_K^{1/2}|}$ (blue dashed line) and $\frac{S_K^{\text{test}}-S_K^{\alpha}}{|S_K^{1/2}|}$ (green dashed line) at each point on the cumulative sum curve. These values show the percentage improvement for each method if only the first K fastest `MatVec` times were included in the dataset for each method. The dynamic α and MPI test method consistently show up to 10% improvement in the overall performance for both methods over fixed α.

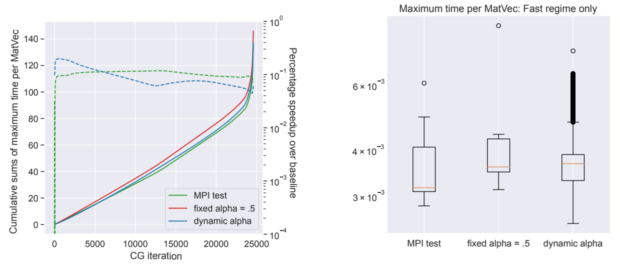

Figure 11: Cumulative sums of the *maximum* time per `MatVec` for the CG solver (left), where the maximum is taken over all MPI ranks for each parallel `MatVec`. Corresponding box plots (right) summarize the maximum `MatVec` times.

On the right side of Figure 11, we also show boxplots of the $T_k^{\max}$, which are clustered using Algorithm 1 with two regimes. The fast regime data also show better performance for the dynamic α method and the MPI test method compared with the fixed α method. In particular, we note that the fastest `MatVec` times are improved for the dynamic α method and the MPI test method compared with fixed α. This supports our conclusion from the cumulative sum plot, that the dynamic α method and the MPI test method reduce the level of the fastest `MatVec` times. Since these low-latency `MatVec` operations comprise a significant portion of all `MatVec` operations, we expect the average overall runtime of the CG solver to be improved by the new methods.

4.4 Frontera – Coarse granularity performance: maximum time per MPI rank

Our last experiments show performance results on the Frontera supercomputer. In Figure 12, we show $T_k^{\max}$ for the three methods. For this experiment, based on a visual inspection of the data we again chose to perform the method of Section 3.1 with two regimes instead of three. The `MatVecs` for each time series are clustered into fast (blue) and slow (red) regimes. We observe an extreme slowdown event for the baseline method (middle panel). The slowdown makes it impossible to directly compare the overall computation times between different algorithms. We experienced this type of extended slowdown for repeatedly for different jobs on Frontera.

In the top panel of Figure 13, we show cumulative sums of the $T_k^{\max}$ times for the three methods (analogous to Section 4.3); the Kth point on each of the curves is the sum of the fastest K `MatVec` times, while the dotted lines show the percent improvement of the new methods over the baseline. Because of the slowdown event, the right tail of the fixed $\alpha = \frac{1}{2}$ curve has a sharp increase, while the MPI test and

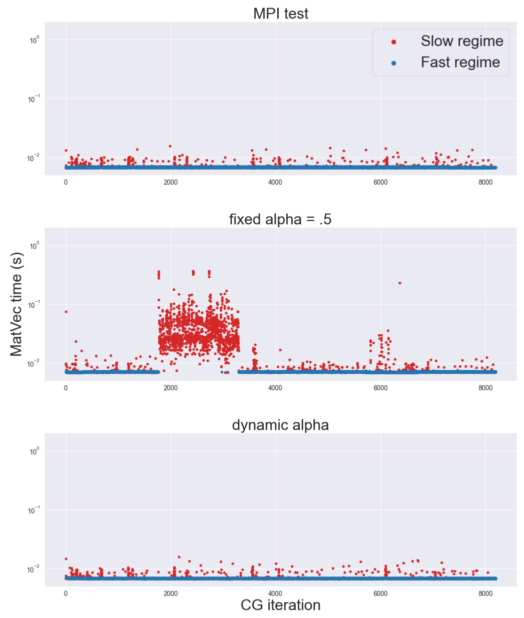

Figure 12: `MatVec` times $T_k^{\max}$ for each of 8192 iterations of the CG solver run on a Poisson problem using three different methods of communication hiding: MPI_Test (top), fixed $\alpha = \frac{1}{2}$ (middle), and dynamic α (bottom).

dynamic α curves are nearly straight lines. The increase for the $\alpha = \frac{1}{2}$ is similar to what we observed in the Anvil data in Section 4.3 for all three methods. The overall runtimes (i.e., the right endpoint of each curve) cannot be directly compared. In contrast, the bottom panel shows the fast regime data only; in this case we can see clearly a 3-5% improvement in runtime. The mean runtime for the fast regime shows a 3.5% improvement for the dynamic α method and a 4.2% improvement for the MPI test method over the $\alpha = \frac{1}{2}$ method.

Remark 2. *The observed network latency patterns differ significantly between Anvil and Frontera (compare, e.g., Figure 6 and Figure 12). While it is beyond the scope of this work to analyze the exact reasons for timing variability on different machines (and, indeed, this work takes the perspective that the causes of performance variability are typically opaque to users), we hope to document such differences and compare the performance of our communication hiding methods on a wider variety of machines in the future.*

5 Conclusions

We document the challenges of analyzing application performance in practice under strong performance variability of the HPC system. We demonstrate that the analysis of timing data in the presence of strong system variabilities is possible with a careful choice of statistical methods. Through the

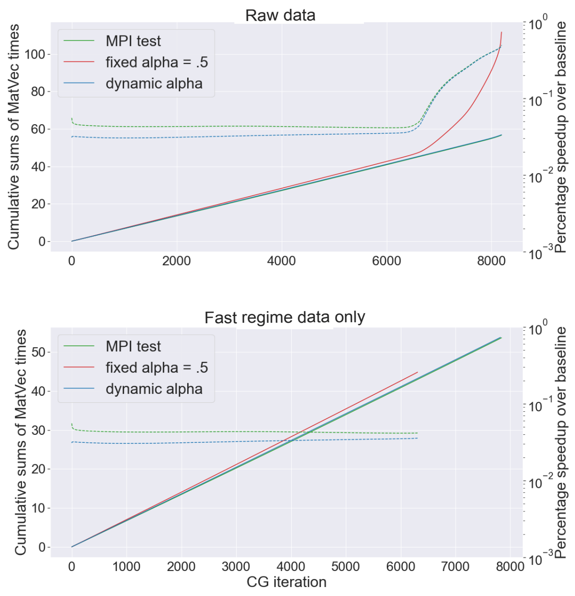

Figure 13: Cumulative sums of raw `MatVec` times (top) and fast regime times only (bottom) over 8192 iterations of the CG solver run on a Poisson problem.

use of unsupervised learning with hidden Markov models and mixture models, we are able to realiably separate the performance of numerical methods from machine noise. We apply our new methodology to sequences of `MatVec` timings on the parallel processes of an HPC system; however, the methodology we introduce can be applied to any application where users can add timing commands, especially parallel applications with repeatedly called subroutines. In the future, we will use our statistical tools to model and simulate complex time-dependent MPI latencies of HPC systems, allowing us to artificially inject and benchmark different latency scenarios that appear on real-world systems.

We have also proposed two new communication hiding methods for matrix-free finite element operators. Only due to our new statistical tools are we able to successfully analyze the high-frequency timing measurements that are otherwise polluted by the performance variablity of HPC systems. With the current trends in HPC hardware architectures, we expect the statistical analysis that we have performed here to be essential for advancements of numerical methods.

In future work, we will consider the effect of allowing the splitting parameter α to vary more rapidly across `MatVec` applications to be able to adapt to higher-frequency variations in communication patterns. Furthermore, we want to analyze individual `MatVecs` in complex iterations such as multigrid-preconditoned Krylov methods and implicit solvers for block systems that require block preconditioning [47, 48]. This likely requires the use of different splitting parameters α for each phase of the preconditioner in a single iteration.

Finally, while the present work has studied one representative configuration of problem size and number of MPI ranks, we will extend our statistical tools and communication hiding to scalability studies on a variety of different machines.

Acknowledgments

This research used resources provided by the Rosen Center for Advanced Computing (RCAC) at Purdue University. and the Texas Advanced Computing Center (TACC) at The University of Texas at Austin.

This research was partially supported by NSF grants OAC-2139536 and EAR-2343865.

This work was authored by the National Renewable Energy Laboratory, operated by Alliance for Sustainable Energy, LLC, for the U.S. Department of Energy (DOE) under Contract No. DE-AC36-08GO28308. This work is supported by the U.S. Department of Energy Office of Advanced Scientific Computing Research through the Scientific Discovery through Advanced Computing (SciDAC) program and through the FASTMath Institute.

References

[1] Laksono Adhianto, Sinchan Banerjee, Mike Fagan, Mark Krentel, Gabriel Marin, John Mellor-Crummey, and Nathan R Tallent. 2010. HPCToolkit: Tools for performance analysis of optimized parallel programs. *Concurrency and Computation: Practice and Experience* 22, 6 (2010), 685–701.

[2] Anthony Agelastos, Benjamin Allan, Jim Brandt, Paul Cassella, Jeremy Enos, Joshi Fullop, Ann Gentile, Steve Monk, Nichamon Naksinehaboon, Jeff Ogden, et al. 2014. The lightweight distributed metric service: a scalable infrastructure for continuous monitoring of large scale computing systems and applications. In *SC'14: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 154–165.

[3] Hirotugu Akaike. 1974. A new look at the statistical model identification. *IEEE Transactions on Automatic Control* 19, 6 (1974), 716–723.

[4] Burak Aksar, Efe Sencan, Benjamin Schwaller, Omar Aaziz, Vitus J Leung, Jim Brandt, Brian Kulis, Manuel Egele, and Ayse K Coskun. 2024. Runtime Performance Anomaly Diagnosis in Production HPC Systems Using Active Learning. *IEEE Transactions on Parallel and Distributed Systems* (2024).

[5] Burak Aksar, Yijia Zhang, Emre Ates, Benjamin Schwaller, Omar Aaziz, Vitus J Leung, Jim Brandt, Manuel Egele, and Ayse K Coskun. 2021. Proctor: A semi-supervised performance anomaly diagnosis framework for production HPC systems. In *High Performance Computing: 36th International Conference, ISC High Performance 2021, Virtual Event, June 24–July 2, 2021, Proceedings 36*. Springer, 195–214.

[6] Daniel Arndt, Wolfgang Bangerth, Denis Davydov, Timo Heister, Luca Heltai, Martin Kronbichler, Matthias Maier, Jean-Paul Pelteret, Bruno Turcksin, and David Wells. 2021. The deal.II finite element library: Design, features, and insights. *Computers & Mathematics with Applications* 81 (2021), 407–422.

[7] Behnaz Arzani, Selim Ciraci, Boon Thau Loo, Assaf Schuster, and Geoff Outhred. 2016. Taking the blame game out of data centers operations

with netpoirot. In *Proceedings of the 2016 ACM SIGCOMM Conference*. 440–453.

[8] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Steven Benson, Jed Brown, Peter Brune, Kris Buschelman, Emil Constantinescu, Lisandro Dalcin, Alp Dener, Victor Eijkhout, Jacob Faibussowitsch, William D. Gropp, Václav Hapla, Tobin Isaac, Pierre Jolivet, Dmitry Karpeev, Dinesh Kaushik, Matthew G. Knepley, Fande Kong, Scott Kruger, Dave A. May, Lois Curfman McInnes, Richard Tran Mills, Lawrence Mitchell, Todd Munson, Jose E. Roman, Karl Rupp, Patrick Sanan, Jason Sarich, Barry F. Smith, Hansol Suh, Stefano Zampini, Hong Zhang, Hong Zhang, and Junchao Zhang. 2024. *PETSc/TAO Users Manual.* Technical Report ANL-21/39 - Revision 3.22. Argonne National Laboratory. doi:10.2172/2205494

[9] Wolfgang Bangerth, Ralf Hartmann, and Guido Kanschat. 2007. deal.II -— A general-purpose object-oriented finite element library. *ACM Transactions on Mathematical Software (TOMS)* 33, 4 (2007), 24–es.

[10] Leonard E. Baum and J. A. Eagon. 1967. An inequality with applications to statistical estimation for probabilistic functions of Markov processes and to a model for ecology. *Bull. Amer. Math. Soc.* 73, 6 (1967), 360–363.

[11] Leonard E Baum and Ted Petrie. 1966. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics* 37, 6 (1966), 1554–1563.

[12] Abhinav Bhatele, Kathryn Mohror, Steven H Langer, and Katherine E Isaacs. 2013. There goes the neighborhood: performance degradation due to nearby jobs. In *SC'13: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*. 1–12.

[13] Monowar H Bhuyan, DK Bhattacharyya, and Jugal K Kalita. 2011. NADO: network anomaly detection using outlier approach. In *Proceedings of the 2011 International Conference on Communication, Computing & Security*. 531–536.

[14] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B Woodard, and Hans Andersen. 2010. Fingerprinting the datacenter: automated classification of performance crises. In *Proceedings of the 5th European conference on Computer systems*. 111–124.

[15] Andrea Borghesi, Antonio Libri, Luca Benini, and Andrea Bartolini. 2019. Online anomaly detection in HPC systems. In *2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 229–233.

[16] Carsten Burstedde, Lucas C. Wilcox, and Omar Ghattas. 2011. `p4est`: Scalable Algorithms for Parallel Adaptive Mesh Refinement on Forests of Octrees. *SIAM Journal on Scientific Computing* 33, 3 (2011), 1103–1133. doi:10.1137/100791634

[17] Sudheer Chunduri, Taylor Groves, Peter Mendygral, Brian Austin, Jacob Balma, Krishna Kandalla, Kalyan Kumaran, Glenn Lockwood, Scott Parker, Steven Warren, et al. 2019. GPCNeT: Designing a benchmark suite for inducing and measuring contention in HPC networks. In *SC'19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–33.

[18] Sudheer Chunduri, Kevin Harms, Scott Parker, Vitali Morozov, Samuel Oshin, Naveen Cherukuri, and Kalyan Kumaran. 2017. Run-to-run variability on Xeon Phi based Cray XC systems. In *SC'17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–13.

[19] Arthur P Dempster, Nan M Laird, and Donald B Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)* 39, 1 (1977), 1–22.

[20] Michel O. Deville, Paul F. Fischer, and Ernest H. Mund. 2002. *High-order methods for incompressible fluid flow.* Cambridge Monographs on Applied and Computational Mathematics, Vol. 9. Cambridge University Press, Cambridge. xxviii+499 pages. doi:10.1017/CBO9780511546792

[21] Paul R Eller, Torsten Hoefler, and William Gropp. 2019. Using performance models to understand scalable Krylov solver performance at scale for structured grid problems. In *Proceedings of the ACM International Conference on Supercomputing*. 138–149.

[22] Kurt B Ferreira, Patrick Bridges, and Ron Brightwell. 2008. Characterizing application sensitivity to OS interference using kernel-level noise injection. In *SC'08: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing*. IEEE, 1–12.

[23] Paul Fischer, Stefan Kerkemeier, Misun Min, Yu-Hsiang Lan, Malachi Phillips, Thilina Rathnayake, Elia Merzari, Ananias Tomboulides, Ali Karakus, Noel Chalmers, and Tim Warburton. 2022. NekRS, a GPU-accelerated spectral element Navier–Stokes solver. *Parallel Comput.* 114 (2022), 102982.

[24] Paul Fischer, James Lottes, David Pointer, and Andrew Siegel. 2008. Petascale algorithms for reactor hydrodynamics. In *Journal of Physics: Conference Series*, Vol. 125. IOP Publishing, 012076.

[25] Paul Fischer, James Lottes, and Henry Tufo. 2007. *Nek5000.* Technical Report. Argonne National Laboratory (ANL), Argonne, IL (United States).

[26] Paul Fischer, Misun Min, Thilina Rathnayake, Som Dutta, Tzanio Kolev, Veselin Dobrev, Jean-Sylvain Camier, Martin Kronbichler, Tim Warburton, Kasia Świrydowicz, et al. 2020. Scalability of high-performance PDE solvers. *The International Journal of High Performance Computing Applications* 34, 5 (2020), 562–586.

[27] Paul F Fischer, Katherine Heisey, and Misun Min. 2015. Scaling limits for PDE-based simulation. In *22nd AIAA Computational Fluid Dynamics Conference*. 3049.

[28] G David Forney. 1973. The Viterbi algorithm. *Proc. IEEE* 61, 3 (1973), 268–278.

[29] Ryan Grant, Kevin Pedretti, and Ann C Gentile. 2014. *Overtime: A Benchmark for Analyzing Performance Variation due to Network Interference.* Technical Report. Sandia National Lab.(SNL-NM), Albuquerque, NM (United States).

[30] Taylor Groves, Yizi Gu, and Nicholas J Wright. 2017. Understanding performance variability on the aries dragonfly network. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 809–813.

[31] Nentawe Gurumdimma, Arshad Jhumka, Maria Liakata, Edward Chuah, and James Browne. 2016. Crude: Combining resource usage data and error logs for accurate error detection in large-scale distributed systems. In *2016 IEEE 35th Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 51–60.

[32] Torsten Hoefler and Roberto Belli. 2015. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–12.

[33] Torsten Hoefler, Timo Schneider, and Andrew Lumsdaine. 2010. Characterizing the influence of system noise on large-scale applications by simulation. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 1–11.

[34] Shi Jin, Zhaobo Zhang, Krishnendu Chakrabarty, and Xinli Gu. 2016. Accurate anomaly detection using correlation-based time-series analysis in a core router system. In *2016 IEEE International Test Conference (ITC)*. IEEE, 1–10.

[35] Stefan Kerkemeier and et al. 2024. gslib v1.0.9 – Sparse communication library. https://github.com/Nek5000/gslib (Accessed: Dec. 11, 2024).

[36] Jannis Klinkenberg, Christian Terboven, Stefan Lankes, and Matthias S Müller. 2017. Data mining-based analysis of HPC center operations. In *2017 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 766–773.

[37] Martin Kronbichler, Timo Heister, and Wolfgang Bangerth. 2012. High accuracy mantle convection simulation through modern numerical methods. *Geophysical Journal International* 191, 1 (2012), 12–29.

[38] Martin Kronbichler and Katharina Kormann. 2012. A generic interface for parallel cell-based finite element operator application. *Computers & Fluids* 63 (2012), 135–147.

[39] Zhiling Lan, Ziming Zheng, and Yawei Li. 2009. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems* 21, 2 (2009), 174–187.

[40] Nikolay Laptev, Saeed Amizadeh, and Ian Flint. 2015. Generic and scalable framework for automated time-series anomaly detection. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 1939–1947.

[41] Elia Merzari, Steven Hamilton, Thomas Evans, Misun Min, Paul Fischer, Stefan Kerkemeier, Jun Fang, Paul Romano, Yu-Hsiang Lan, Malachi Phillips, et al. 2023. Exascale multiphysics nuclear reactor simulations for advanced designs. In *SC'23: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–11.

[42] Martin Molan, Andrea Borghesi, Daniele Cesarini, Luca Benini, and Andrea Bartolini. 2023. RUAD: Unsupervised anomaly detection in HPC systems. *Future Generation Computer Systems* 141 (2023), 542–554.

[43] Peter Munch, Timo Heister, Laura Prieto Saavedra, and Martin Kronbichler. 2023. Efficient distributed matrix-free multigrid methods on locally refined meshes for FEM computations. *ACM Transactions on Parallel Computing* 10, 1 (2023), 1–38.

[44] Vinod Nair, Ameya Raul, Shwetabh Khanduja, Vikas Bahirwani, Qihong Shao, Sundararajan Sellamanickam, Sathiya Keerthi, Steve Herbert, and Sudheer Dhulipalla. 2015. Learning a hierarchical monitoring system for detecting and diagnosing service issues. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 2029–2038.

[45] Lawrence R Rabiner. 1989. A tutorial on hidden Markov models and selected applications in speech recognition. *Proc. IEEE* 77, 2 (1989), 257–286.

[46] Johann Rudi, A. Cristiano I. Malossi, Tobin Isaac, Georg Stadler, Michael Gurnis, Peter W. J. Staar, Yves Ineichen, Costas Bekas, Alessandro Curioni, and Omar Ghattas. 2015. An Extreme-Scale Implicit Solver for Complex PDEs: Highly Heterogeneous Flow in Earth's Mantle. In *SC'15: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. ACM, Article 5, 12 pages. doi:10.1145/2807591.2807675

[47] Johann Rudi, Yu-Hsuan Shih, and Georg Stadler. 2020. Advanced Newton Methods for Geodynamical Models of Stokes Flow with Viscoplastic Rheologies. *Geochemistry, Geophysics, Geosystems* 21, 9 (2020). doi:10.1029/2020GC009059

[48] Johann Rudi, Georg Stadler, and Omar Ghattas. 2017. Weighted BFBT Preconditioner for Stokes Flow Problems with Highly Heterogeneous Viscosity. *SIAM Journal on Scientific Computing* 39, 5 (2017), S272–S297. doi:10.1137/16M108450X

[49] P. Sanan, S.M. Schnepp, and D.A. May. 2016. Pipelined, Flexible Krylov Subspace Methods. *SIAM Journal on Scientific Computing* 38, 5 (2016), C441–C470. doi:10.1137/15M1049130 arXiv:https://doi.org/10.1137/15M1049130

[50] David Skinner and William Kramer. 2005. Understanding the causes of performance variability in HPC workloads. In *IEEE International. 2005 Proceedings of the IEEE Workload Characterization Symposium, 2005.* IEEE, 137–149.

[51] Ying Sun and Marc G Genton. 2011. Functional boxplots. *Journal of Computational and Graphical Statistics* 20, 2 (2011), 316–334.

[52] Philip Taffet and John Mellor-Crummey. 2019. Understanding congestion in high performance interconnection networks using sampling. In *SC'19: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 1–24.

[53] Henry M Tufo and Paul F Fischer. 1999. Terascale spectral element algorithms and implementations. In *SC'99: Proceedings of the 1999 ACM/IEEE Conference on Supercomputing*. 68–es.

[54] Ozan Tuncer, Emre Ates, Yijia Zhang, Ata Turk, Jim Brandt, Vitus J Leung, Manuel Egele, and Ayse K Coskun. 2017. Diagnosing performance variations in HPC applications using machine learning. In *High Performance Computing: 32nd International Conference, ISC High Performance 2017, Frankfurt, Germany, June 18–22, 2017, Proceedings 32.* Springer, 355–373.

[55] Ozan Tuncer, Emre Ates, Yijia Zhang, Ata Turk, Jim Brandt, Vitus J Leung, Manuel Egele, and Ayse K Coskun. 2018. Online diagnosis of performance variation in HPC systems using machine learning. *IEEE Transactions on Parallel and Distributed Systems* 30, 4 (2018), 883–896.

[56] Robert Underwood, Jason Anderson, and Amy Apon. 2018. Measuring Network Latency Variation Impacts to High Performance Computing Application Performance. In *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering* (Berlin, Germany) *(ICPE '18)*. Association for Computing Machinery, New York, NY, USA, 68––79. doi:10.1145/3184407.3184427

[57] Ingmar Visser and Maarten Speekenbrink. 2021. hmmr v1.0-0 – Mixture and Hidden Markov Models with R. doi:10.32614/CRAN.package.hmmr https://CRAN.R-project.org/package=hmmr (Accessed: Dec. 15, 2024).

[58] Andrew J. Viterbi. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory* 13, 2 (1967), 260–269.

[59] Hannes Weisbach, Balazs Gerofi, Brian Kocoloski, Hermann Härtig, and Yutaka Ishikawa. 2018. Hardware performance variation: A comparative study using lightweight kernels. In *High Performance Computing: 33rd International Conference, ISC High Performance 2018, Frankfurt, Germany, June 24-28, 2018, Proceedings 33.* Springer, 246–265.

[60] Xu Yang, John Jenkins, Misbah Mubarak, Robert B Ross, and Zhiling Lan. 2016. Watch out for the bully! job interference study on dragonfly network. In *SC'16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. IEEE, 750–760.

[61] Li Yu and Zhiling Lan. 2015. A scalable, non-parametric method for detecting performance anomaly in large scale computing. *IEEE Transactions on Parallel and Distributed Systems* 27, 7 (2015), 1902–1914.

[62] Xiao Zhang, Fanjing Meng, Pengfei Chen, and Jingmin Xu. 2016. Taskinsight: A fine-grained performance anomaly detection and problem locating system. In *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*. IEEE, 917–920.

[63] Yijia Zhang, Taylor Groves, Brandon Cook, Nicholas J Wright, and Ayse K Coskun. 2020. Quantifying the impact of network congestion on application performance and network metrics. In *2020 IEEE International Conference on Cluster Computing (CLUSTER)*. IEEE, 162–168.