International Journal of
HIGH PERFORMANCE
COMPUTING APPLICATIONS

Research Paper

The International Journal of High Performance Computing Applications
1–13
© The Author(s) 2020
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/1094342020966835
journals.sagepub.com/home/hpc
SAGE

Understanding performance variability in standard and pipelined parallel Krylov solvers

Hannah Morgan[1], Patrick Sanan[2], Matthew Knepley[3] and Richard Tran Mills[1]

Abstract
In this work, we collect data from runs of Krylov subspace methods and pipelined Krylov algorithms in an effort to understand and model the impact of machine noise and other sources of variability on performance. We find large variability of Krylov iterations between compute nodes for standard methods that is reduced in pipelined algorithms, directly supporting conjecture, as well as large variation between statistical distributions of runtimes across iterations. Based on these results, we improve upon a previously introduced nondeterministic performance model by allowing iterations to fluctuate over time. We present our data from runs of various Krylov algorithms across multiple platforms as well as our updated non-stationary model that provides good agreement with observations. We also suggest how it can be used as a predictive tool.

Keywords
Krylov, pipelining, noise, performance model, GMRES, PGMRES, BiCGStab

1 Introduction

Coherent performance models are important to describe and evaluate algorithms on modern architectures. With good models, HPC users can pick the appropriate method and algorithmic parameters for their application to run in a complicated computing environment. Furthermore, models are important in evaluating HPC systems and moving forward with new architecture and algorithm design.

The increasing complexity of computing environments has introduced many sources of run-to-run variability at different levels of HPC systems. For example, an operating system can interrupt processors at any time for its activities, causing detours in computations and degrading performance (Ferreira et al., 2008; Hoefler et al., 2010). Inter-job contention for shared resources such as network bandwidth, routers, and links is another source of variability that can affect application performance (Chunduri et al., 2017; Parker et al., 2017). Developing coherent performance models in the presence of variability is difficult (Beckman et al., 2006; Hoefler et al., 2007), particularly between systems.

In this work, we advocate for casting performance models in a nondeterministic paradigm that more accurately reflects algorithm execution in noisy HPC environments. Employing stochastic performance models for algorithms that run on HPC systems could reflect a more realistic computing scenario since many detours and sources of variability are unpredictable. Some work has been done using stochastic models to predict performance in HPC environments. For example, Agarwal et al. (2005) studied the impact of different noise distributions on a single collective operation and Seelam et al. (2010) modeled operating system "jitter" as random variables in order to compute computational slowdown in the presence of noise.

We are interested in modeling standard Krylov subspace methods (Saad, 1996; van der Vorst, 2003) and pipelined variants often used in large-scale simulations to solve sparse systems of equations. Highly synchronous applications such as those that use Krylov subspace methods are vulnerable to performance degradation induced by variability at different machine levels. Because each iteration is performed in lockstep, an operating system interrupt (or any other source of slowdown) that delays a single

[1] Argonne National Laboratory, Lemont, IL, USA
[2] ETH Zurich, Zurich, Switzerland
[3] University at Buffalo, Buffalo, NY, USA

Corresponding author:
Hannah Morgan, Mathematics and Computer Science, Argonne National Laboratory, Argonne, Lemont, IL, USA.
Email: hmmorgan@uchicago.edu

processor can cause others to be delayed as well. The cost of communication can vary considerably between iterations and runs on modern machines due to network contention or other factors. This can significantly degrade method performance because of the global reduction operations used to compute vector norms and inner products. Motivated by mitigating latency associated with global communication, algebraically equivalent pipelined algorithms of Krylov subspace methods have been introduced (Chronopoulos and Gear, 1989; Ghysels and Vanroose, 2014; Ghysels et al., 2013; Jacques et al., 1999; Strzodka and Göddeke, 2006; Sturler and van der Vorst, 1995). Pipelined algorithms rearrange computations so that it is possible to hide some of the cost of global communication with local computation, at the cost of increased computation, storage, and degraded numerical stability.

We characterize the performance of standard Krylov subspace methods and pipelined algorithms using experimental data to refine the performance model introduced (see Morgan et al., 2016), which used random variables to model iteration times. Such models and data are useful because pipelined Krylov algorithms tend to become more useful in situations where performance data is more difficult to directly gather: with large node counts on congested supercomputers with unpredictable local processing times. We intentionally maintain a data-driven approach, focusing on a simple stochastic model that nevertheless captures a key property of pipelined Krylov algorithms in the context of increasingly-complex computing systems. This is to be strongly contrasted with traditional performance modeling which emphasizes describing best-case scenarios (Gropp and Lusk, 1999), attainable speedup, and detailed measurements of presumed-to-be-relevant performance characteristics.

Data is found to directly support the conjecture that pipelined algorithms effectively smooth out variability which in and of itself can give speedup over standard Krylov subspace methods. This demonstrates that in addition to helping to address the scalability bottleneck associated with global reductions, pipelined Krylov algorithms can, by relaxing data dependencies, offer increased performance even when reductions are modeled as simple barriers. This points to these methods being useful in a much wider range of applications, such as those with smaller problem sizes but more variable local times per iteration, or on systems with extremely fast networks that are currently far away from a reduction latency scaling bottleneck.

In this paper, we begin by reviewing the performance models introduced in Morgan et al. (2016) for Krylov subspace methods and pipelined algorithms in the presence of noise. Next we examine experimental results from runs of these algorithms and refine the performance models based on insights from the data. Specifically, we intend to show the need for non-stationary models. Throughout this work, we use the term "non-stationary" to mean a process (in our case, iterations of a Krylov algorithm) whose probability distribution changes with shifts in time. We employ our stochastic performance models and show that the updated model is in close agreement with reality. Then we present more experimental data and suggest ways to perform a priori performance estimates. Finally, we conclude our work.

2 Stochastic model review

A standard Krylov method is modeled as a set of P processes repeatedly performing local computation and global communication. Because of the synchronizations in each of iteration the total time for each iteration is the time given by the slowest processor. Let p index the P processors and k index the K iterations. Then, the total time T for a Krylov method is modeled as

$$T = \sum_k \max_p T_p^k \tag{1}$$

where T_p^k is the time for iteration k on process p. Since this work can be interrupted by operating system noise or other interference, the iteration times T_p^k might fluctuate over steps and processors. To model this nondeterministic behavior, we let the iterates be random variables and ask for the expected total runtime

$$E[T] = \sum_k E\left[\max_p \mathcal{T}_p^k\right] \tag{2}$$

If the iterates $\mathcal{T}_p^k$ are identical and independent of processor (p) and stationary in time (k), then we can compute the expected runtime of a Krylov method using the integral expression

$$E[T] = KP\int_{-\infty}^{\infty} xF(x)^{P-1}f(x)\,\mathrm{d}x \tag{3}$$

where the random variables $\mathcal{T}_p^k$ are drawn from a distribution with probability density function (pdf) $f(x)$ and cumulative distribution function (cdf) $F(x)$.

Similarly, we model a pipelined Krylov algorithm as a set of P communicating processors. Pipelined algorithms employ split-phase, non-blocking collectives that do not require global synchronizations. Rather, the collective operation is first initialized, say with `MPI_Iallreduce()`, and later finalized with `MPI_Wait()` so that useful work can continue between initialization and finalization. Provided no processor falls too far behind, the total time for a pipelined algorithm T' is the time for the slowest processor to perform all the given work

$$T' = \max_p \sum_k T_p^k \tag{4}$$

Because of detours, we do not expect the iteration times to be deterministic. Again, we ask for the expected total run time

$$E[T'] = E\left[\max_p \sum_k \mathcal{T}_p^k\right] \tag{5}$$

letting the iterates be random variables drawn from a distribution. If the iterates $\mathcal{T}_p^k$ are identically distributed with finite mean and variance, independent of process (p) and stationary in time (k), then, in the limit of large K, $\frac{1}{K}\sum_k \mathcal{T}_p^k \to \mu$ where μ is the mean of the underlying iteration time distribution with pdf $f(x)$ and cdf $F(x)$. So we have an expression to compute the expected total time for a pipelined algorithm

$$E[T^{'}] \to K\mu. \tag{6}$$

For a more thorough explanation of the arguments presented in this section, see Morgan et al. (2016). In the next section we present fine-grained data collected from runs of Krylov methods and pipelined algorithms. This gives us insight into how detours affect these simulations, which assumptions above are realistic, and how the models might be refined.

3 Experimental results

Here we present the results from experiments conducted on the Cray XC40, Theta, at the Argonne Leadership Computing Facility. We choose to run two programs with contrasting characteristics. While this is a limited representation the results are informative. Theta compute nodes each contain a single Xeon Phi Knights Landing (KNL) CPU (Sodani, 2015) with 64 cores, and are connected by a Cray Aries interconnect in a dragonfly topology (Alverson et al., 2012). We use the Portable, Extensible Toolkit for Scientific Computation (PETSc) (Balay et al., 2018a, 2018b) version 3.10 to simulate the solution to problems with varying characteristics such as sparsity pattern. One MPI rank per core is used throughout and we boot the nodes in quadrant clustering mode and flat memory mode, which exposes the on-die, high-bandwidth Multi-Channel DRAM (MCDRAM) as a user-addressable memory, rather than treating it as an L3 cache. We collect iteration times $\mathcal{T}_p^k$ by inserting calls to `MPI_Wtime()` at the beginning and end of each Krylov cycle in PETSc's scalable linear equation solvers (KSP) component's GMRES (generalized minimal residual method) and PGMRES (pipelined GMRES) algorithms (Ghysels et al., 2013). Since changes and upgrades to HPC machines can affect performance, we note that experiments presented in this paper were performed in autumn 2018 with Cray-MPICH 7.7.3 and built with the Cray C compiler wrapper.

3.1 One-dimensional Laplacian

PETSc KSP tutorial ex23[1] is a one-dimensional finite-difference discretization of the Laplacian, resulting in a simple tridiagonal system. A simple matrix will subtract the effects of irregular communication in sparse matrix–vector products by removing the need for all but nearest neighbor communication. We show the results from a simulation with a more complicated sparsity pattern in the next section. We run ex23 with GMRES and PGMRES using 8192 MPI processes with 10^6 unknowns and a Jacobi preconditioner. Throughout these simulations we force

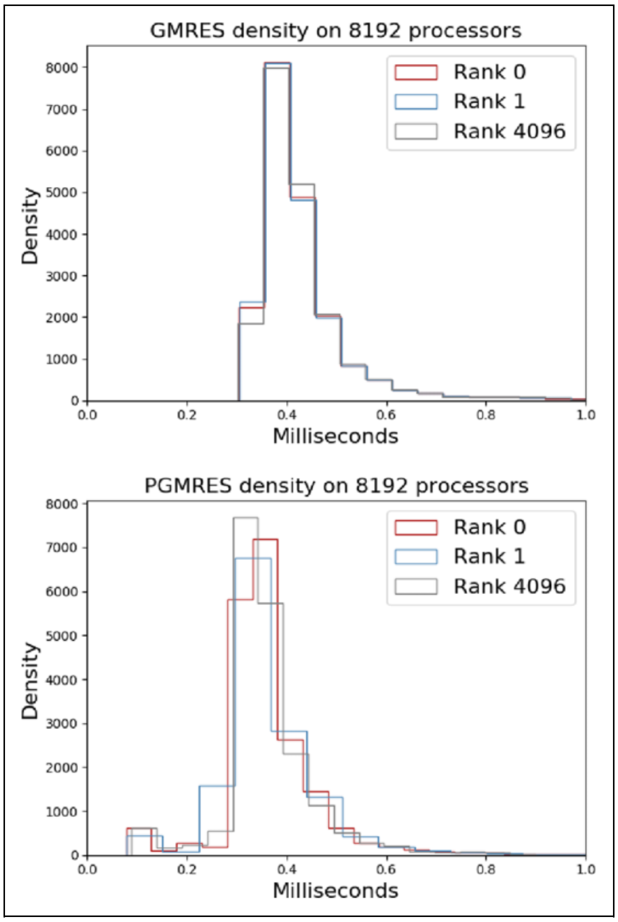

Figure 1. Distribution of iterates $\mathcal{T}_p^k$ for fixed processors.

5000 Krylov iterations. We choose to run our experiments with few degrees of freedom per processor since these schemes are often used in this regime and with a large number of iterations to amass statistics.

The stochastic model expressions (3) and (6) place assumptions on the iterates $\mathcal{T}_p^k$. We check them here to examine the actual performance of the algorithms and to justify use of the models. In Figure 1 we graph the distribution of the iterates for select processors identified by their MPI ranks. The distributions look similar between ranks, implying that the iterations might be identical with respect to process. We make use of the two sample Kolmogorov–Smirnov test to check whether two samples (e.g. iterations from rank 0 and rank 1) are drawn from the same underlying distribution. The Kolmogorov-Smirnov test calculates the distance between the empirical distributions of two samples with empirical distribution functions F_1 of size n and F_2 of size m with

$$D = \sup_x |F_1(x) - F_2(x)| \tag{7}$$

We reject the hypothesis that the samples come from the same distribution with significance level α if

$$D > c(\alpha)\sqrt{\frac{n+m}{nm}} \tag{8}$$

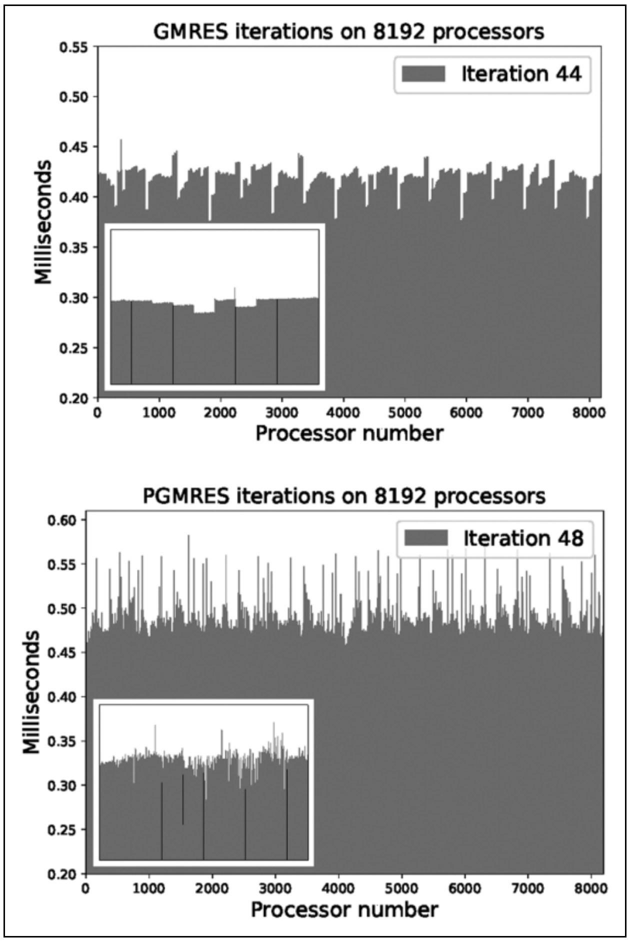

Figure 2. Time for iterations $k = 44$ (GMRES) and $k = 48$ (PGMRES) on process p with inset showing the first 10 nodes.

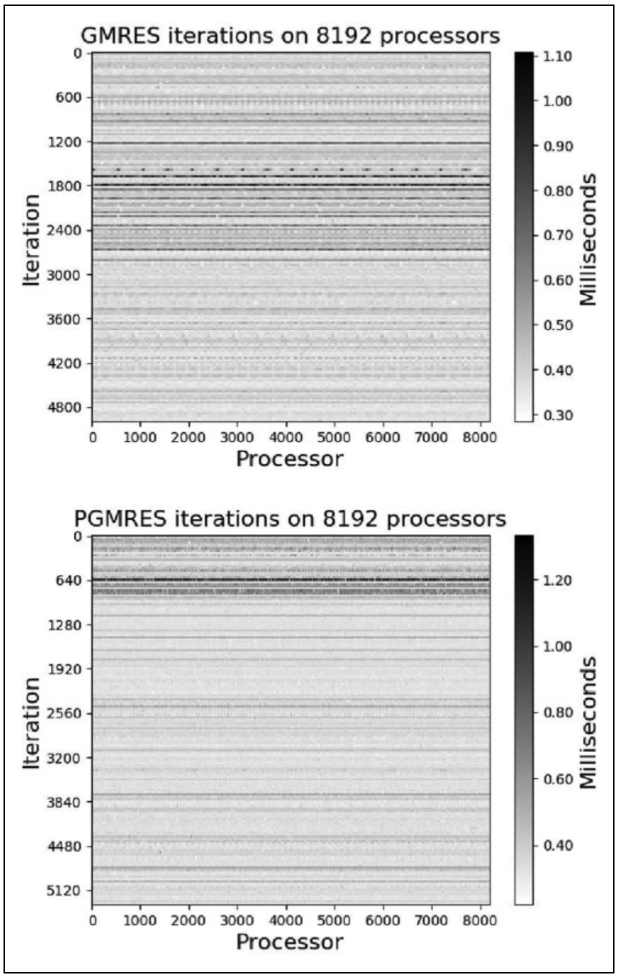

Figure 3. Two-color colormap of iterates $\mathcal{T}_p^k$.

where $c(\alpha)$ is a tabulated value. For GMRES, $n = m = 5000$ and $n = m = 5334$ for PGMRES. We use significance level $\alpha = 0.05$ and SciPy's `stat.ks_2samp` function to calculate the test statistic D. We find that we do not reject that GMRES ranks 0 and 1 come from the same distribution with significance level $\alpha = 0.05$ since $D = 0.002 < 0.024$, the threshold. For PGMRES ranks 0 and 1, $D = 0.088 > 0.023$, so we reject the null hypothesis. On rank 1, we reject none of the $P - 1$ pairs of GMRES ranks and reject 54.5% of PGMRES pairs.

Because it is inefficient to orthogonalize new Krylov vectors against a large basis, GMRES methods are typically implemented to restart after R iterations. When a Krylov subspace method restarts, the basis for the Krylov subspace is discarded and the current solution is used as the initial guess for the next cycle. PETSc implements a pipelined GMRES algorithm given by Ghysels et al. (2013) which adds two iterations in each cycle to "fill the pipeline." Since the PETSc default restart value is 30, 5334 iterations of PGMRES are actually employed instead of 5000. Some of these are very quick and easily identifiable as the bump of very short iterations in Figure 1 (bottom). Still, we compare 30 GMRES iterations to 32 of PGMRES since they compute the same number of intermediate solutions which are, moreover, the same in exact arithmetic.

Figure 2 shows the time for iterations $k = 44$ (GMRES) and $k = 48$ (PGMRES), halfway through the Krylov cycle described before, on process p in order of MPI rank. We include a inset graph that shows the iterations on the first 10 nodes, delineated by black lines. GMRES iterates are nearly constant on a node; each KNL node of 64 processes is clearly visible in Figure 2 (top, inset) so that the iterations are not truly independent in p as we assumed above. The variance of iteration times on node 0 (with MPI ranks 0–63) is $6.3 \cdot 10^{-13}\text{s}^2$. This could be due to the synchronizations in each iteration of GMRES that force processes on a node to perform in lockstep. There is also some periodic behavior over all the nodes. Conversely, PGMRES iterations contain more variability on a node ($1.4 \cdot 10^{-10}\text{s}^2$ on the node with MPI ranks 0–63, three orders of magnitude greater than GMRES), but less between them.

Figure 3 is a two-color colormap of the iterates $\mathcal{T}_p^k$ for all iterations and all processors. Graphed are the iterations that perform the "average" amount of work during each GMRES cycle (GMRES iteration 14mod30 and PGMRES 16mod32) so that each iteration shown performs the same

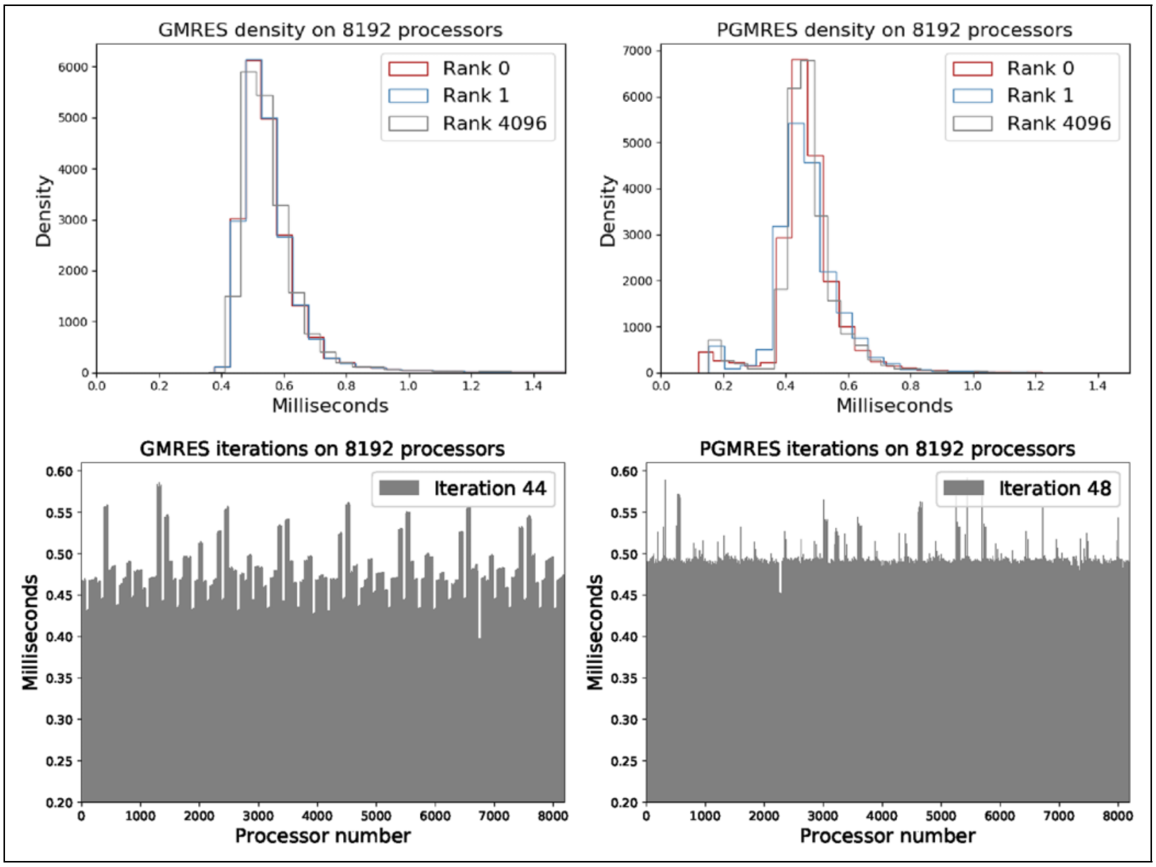

Figure 4. GMRES (left) and PGMRES (right) iterations for PETSc ex48.

computation. Horizontal lines are highly visible, implying that processors perform similarly within a given iteration. The difference between different iterations is also clear; iterate timings fluctuate in time without an obvious pattern. In the absence of an algorithmic reason that some iterations should take longer than others, after taking restarts into account, these differences could be explained by long operating system interruptions, thermal throttling of the CPU, or other external causes. Furthermore, some groups of iterations (GMRES iterations $k \approx 1700 - 2300$ or PGMRES $k \approx 640 - 740$) overall take longer than other iterations.

In the last section we made the simplifying assumption that the iterates were stationary in time so that the sum in (2) could become a product in (3). We account for this non-stationary behavior in the next section.

3.2 Three-dimensional non-Newtonian fluid

PETSc SNES tutorial ex48[2] solves the Blatter-Pattyn hydrostatic equations that model ice sheet flow for glaciers (Pattyn et al., 2008). For the ice, we use a power-law rheology with Glen exponent 3. This generates a much denser system of equations with about 10 times more nonzeros per row than ex23 and requires extensive communication across the machine, resulting in more network traffic and the possibility for more variable communication latency. Again we choose our problem size so that there are 10^6 unknowns, use a Jacobi preconditioner, and force 5000 iterates of the Krylov algorithm. We stop after one non-linear iteration.

We repeat the analysis from before and find qualitatively similar results, shown in Figure 4. Using the Kolmogorov-Smirnov test again we find that we do not reject the assumption that GMRES iterates from ranks 0 and 1 come from the same distribution since $D = 0.003 < 0.024$ and we reject that PGMRES iterates come from the same distribution since $D = 0.078 > 0.023$. The iterates on a fixed process p look like they could be from the same family of distributions with different parameters as those in the previous subsection.

GMRES iterates again appear highly dependent on the node with node 0 variance $4.3 \cdot 10^{-13}\mathrm{s}^2$. The variance on PGMRES node 0 is $8.6 \cdot 10^{-10}\mathrm{s}^2$, larger than before, and there is more variation between PGMRES nodes which looks periodic. This could be explained by the communication induced by the matrix sparsity pattern.

4 Non-stationary performance model

Based on the results from the last section, we move to a non-stationary performance model where the iterates $\mathcal{T}_p^k$ can fluctuate in time. The expected total time for a Krylov method is still given by

$$E[T] = \sum_k E\left[\max_p \mathcal{T}_p^k\right]$$

but we relax the claim that the iterates are stationary in time. In iteration k, the random variables $\mathcal{T}_p^k$ are drawn from a distribution with cdf $F_k(x)$ and pdf $f_k(x)$, which can now fluctuate across steps. Then the expected runtime of a Krylov method is given by

$$\hat{E}[T] = \sum_k P \int_{-\infty}^{\infty} x F_k(x)^{P-1} f_k(x)\,\mathrm{d}x \tag{9}$$

We call this the "non-stationary" performance model and distinguish between $E[T]$ in (3) and $\hat{E}[T]$ here.

Similarly for a pipelined algorithm, we use (5) as before and drop the assumption that the iterates are stationary in time, so that

$$\hat{E}[T'] \rightarrow \sum_k \mu_k \tag{10}$$

where μ_k is the mean of the underlying distribution in iteration k.

The original Krylov model (3) benefits from slight alteration as well. Since the iterates on each node are nearly constant, shown in Figure 2 (left) and Figure 4 (bottom left), they are more suitably modeled by $\hat{P} = P/64$ independent random variables than P independent processors. We make use of this substitution in the next section but it does not seem to strongly affect (9).

5 Performance modeling

In this section, we will deploy our performance models and test them against collected data.

5.1 Computing expected runtimes

The expressions for $E[T]$ and $E[T']$ say that the iterates come from some underlying distribution with pdf $f(x)$, cdf $F(x)$, and mean μ. We will use "bulk statistics" with these performance models, taking all iterations for all processors in aggregate, for the underlying distribution and to characterize the iteration times and machine variation.

To find the functions $f(x)$ and $F(x)$ and mean μ, we use Scipy's `stats` package to fit various analytical distributions to our collected data. For a given continuous distribution and data, the `fit` function returns the maximum likelihood estimation for the shape, location, and scale parameters by maximizing a log-likelihood function. We choose distributions which minimize the sum of squared error between the data and the fit distribution. For the pipelined algorithms, we disregard the two iterations in each cycle that "fill the pipeline" to avoid distribution fitting complications.

Using these well-fitting distributions, we can calculate $E[T]$ and $E[T']$ and compare the expected results to the `KSPSolve` time (the time spent inside the Krylov solver) provided by PETSc's `-log_view` option. For integration in (3) we use Python's `scipy.integrate.quad` function and integrate over the bounds of the data.

To employ the updated models $\hat{E}[T]$ and $\hat{E}[T']$, we again perform distribution fitting using `scipy`, this time for each iteration to find $f_k(x)$, $F_k(x)$, and mean μ_k. GMRES and PGMRES iteration data is shown in Figure 6. It is quite clear that these are far from normal and do not resemble well-known distributions. Because of this, and in the interest of simplicity, we treat runtimes $\mathcal{T}_p^k$ for each iteration k as coming from a uniform distribution with possibly different parameters. Again, integration is performed in each iteration again using `scipy.integrate.quad` and the calculated expected total times are compared to the `KSPSolve` time.

Analytical expressions have been used to bound the maximum of a set of random variables using the sample mean μ_X and standard deviation σ_X (Seelam et al., 2010). Cramer (Cramér, 2016; David and Nagaraja, 2004) bound the maximum of N identical and independent random variables with

$$E[X_{\max(N)}] \leq \mu_X + \frac{\sigma_X(N-1)}{\sqrt{2N-1}} \tag{11}$$

and Bertsimas et al. (2006) bound the maximum of N identical, but not independent random variables by

$$E[X_{\max(N)}] \leq \mu_X + \sigma_X\sqrt{N-1} \tag{12}$$

We will use these to compare with our Krylov performance models.

5.2 Results

Distribution fitting alone gives some insight into algorithm execution. In Figure 5, we show bulk data from our experimental results with the Johnson $\mathrm{S_U}$ distribution, a transformation of the normal distribution with pdf

$$f(x,a,b) = \frac{b}{\sqrt{x^2+1}}\phi(a + b\log(x + \sqrt{x^2+1})) \tag{13}$$

with shape parameters a and b and where ϕ is the normal pdf. Note that there were some other well-fitting distributions, including Non-central Student's T. The bulk data show that in a given method, the iterates $\mathcal{T}_p^k$ are mostly clustered and fairly quick with a fewer, longer, giving the distribution a small tail. In general, PGMRES runtimes were faster than GMRES, consistent with faster overall execution in these examples. Ex48 runtimes are shifted to the right of ex23 and more spread out, implying longer iterations with more variation.

Results for the data presented in our experimental results are shown in Table 1 for PETSc ex23 and ex48.

It is clear that applying the non-stationary performance model $\hat{E}[T]$ to standard Krylov subspace methods gives a large improvement over the stationary model $E[T]$. The stationary model overestimates actual runtimes and the analytical bounds are very loose and not particularly helpful. Both models are good estimates for pipelined algorithms, suggesting that modeling a method without explicit synchronization is more flexible.

In this work we don't compare our results to more algorithm-specific models or those that collect detailed hardware measurements, instead emphasizing a paradigm where the use of simple, general models is sufficient to approximate the execution time for Krylov solvers without collecting low-level information.

6 Predicting runtimes

In the previous section, we showed that our performance models $\hat{E}[T]$ and $\hat{E}[T']$ can reasonably predict execution time when we have the time for iteration k on process p for all k and p.

While bulk statistics appear to be enough to succinctly describe the performance of a pipelined algorithm, they are not a sufficient way to describe a traditional Krylov method. Instead, we will use non-stationary distributions to model each iteration. GMRES and PGMRES distribution data is shown with uniform distributions in Figure 6. Here we see how much the iterations shift in time. They are mostly non-overlapping. We also want to know how the uniform parameters change over time. That is, for each iteration k, the random variables $\mathcal{T}_p^k$ are modeled as following a uniform distribution with pdf f_k and cdf F_k given by

$$f_k(x) = \frac{1}{b_k - a_k}, \quad F_k(x) = \frac{x - a_k}{b_k - a_k} \tag{14}$$

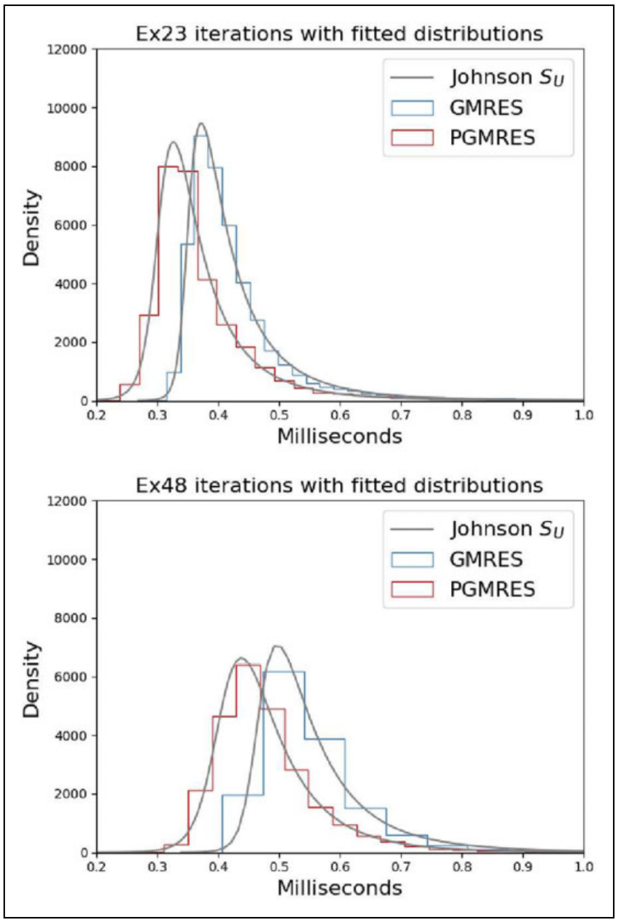

Figure 5. GMRES and PGMRES bulk statistics with fitted distributions for PETSc ex23 (left) and ex48 (right).

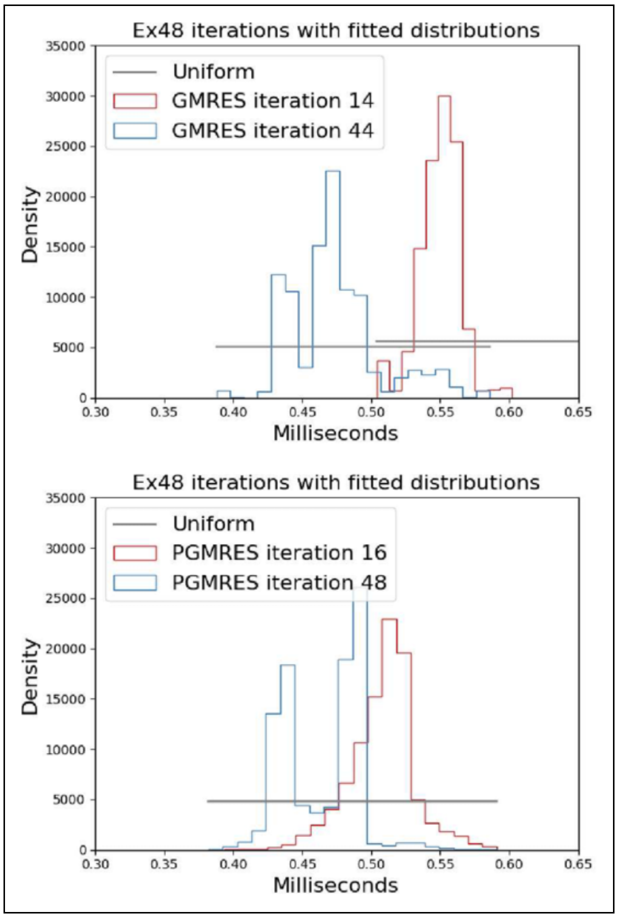

Figure 6. GMRES iterations $k = 14, 44$ and PGMRES $k = 16, 48$ iterations with fitted uniform distributions.

Table 1. Performance model results and bounds for ex23 and ex48 with good fitting distributions.

ex23	KSPSolve	$E_{\mathrm{JohnS_U}}[T]$	$E_{\mathrm{NCT}}[T]$	$\hat{E}_{\mathrm{Unif}}[T]$	Cramer bound	Bertsimas bound
GMRES	2.217	4.831	4.365	2.432	6.35	8.102
PGMRES	2.006	1.865	1.868	1.857		
ex48						
GMRES	2.943	5.716	10.88	3.189	20.59	27.99
PGMRES	2.656	2.413	2.413	2.455		

The uniform distribution can be described with two parameters: the minimum a_k and the span $s_k = b_k - a_k$. We model a Krylov method with K uniform distributions where in each iteration the uniform parameters a_k and s_k are random variables themselves drawn from some distribution. Figure 7 show histograms of the uniform parameters from PETSc ex48 using GMRES and PGMRES with fitted Johnson S_U distribution. This distribution has the pdf (13) and is described by two shape parameters a and b and location and scale parameters `loc` and `scale` in Scipy.

Table 2 shows Johnson S_U parameters for the uniform distribution parameters from Figure 7 and PETSc ex23. In general, they show that, in a given iteration, the fastest processor is generally faster in a PGMRES run, but the runtimes are more spread out. All Johnson S_U distributions are shaped such that they lean left with a tail, some longer than others.

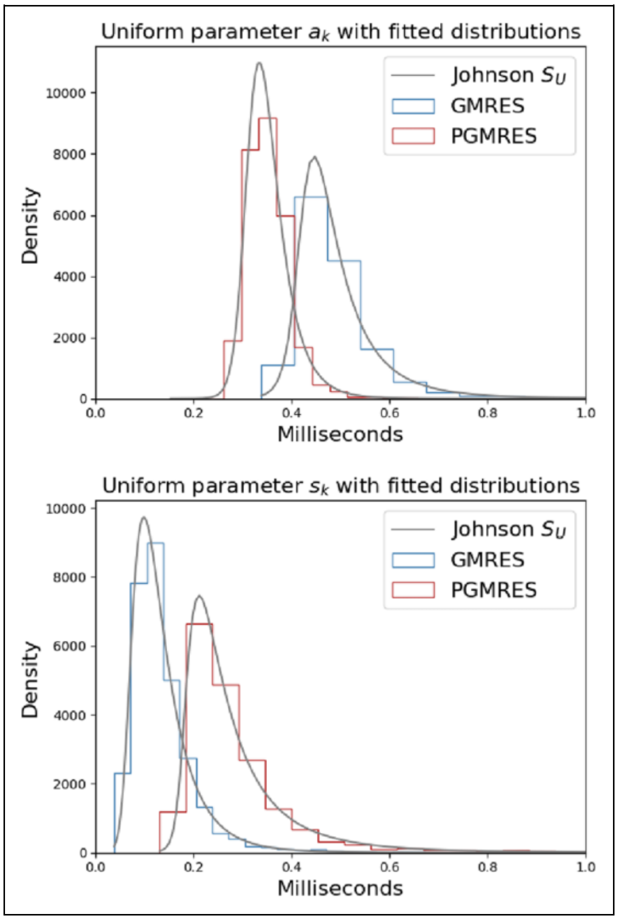

Figure 7. Uniform parameters a_k and s_k from PETSc ex48 with fitted distributions.

We can mimic a Krylov computation by "simulating" the iterations with a Monte Carlo-type approach. We assume that each iteration time $\mathcal{T}_p^k$ is uniformly distributed with parameters a_k and s_k in iteration k

$$\mathcal{T}_p^k \sim \text{Uniform}(a_k, s_k)$$

We further assume that the parameters a_k and s_k are themselves random variables drawn from a Johnson S_U distribution with parameters a, b, `loc`, and `scale`.

We draw random variables a_k and s_k using Scipy's `rvs` function and compute the expected time for iteration k, $\hat{E}[T_k]$, given by

$$\hat{E}[T_k] = \hat{P} \int_{-\infty}^{\infty} x F_k(x)^{\hat{P}-1} f_k(x)\, \mathrm{d}x \tag{15}$$

Repeating for K iterations, we get $\hat{E}[T]$. We simulate a pipelined Krylov computation in the same way, pulling random variables a_k and s_k and computing

$$\hat{E}[T'_k] = \mu_k \tag{16}$$

where μ_k is the mean of $\text{Uniform}(a_k, s_k)$. Table 3 shows the results of this simulation using the Johnson S_U parameters from Table 2. The results are good when we have the computed Johnson S_U parameters. The challenge in general will be to reason about these parameters so that we can make a priori performance estimates for standard Krylov methods and pipelined algorithms.

7 Extension to other algorithms and computing platforms

In the last section, we saw that we can reasonably estimate the execution time of a Krylov method given that we know the Johnson S_U parameters that model a_k and s_k. Many

Table 3. Actual and simulated runtimes in seconds on 8192 processors with 10^6 unknowns.

ex23	GMRES	PGMRES	ex48	GMRES	PGMRES
KSPSolve	2.217	2.006	KSPSolve	2.943	2.656
Simulated	2.416	1.908	Simulated	3.14	2.482

Table 2. Johnson S_U parameters for uniform parameters a_k and s_k.

	Uniform a_k				Uniform s_k			
ex23	*a*	*b*	*loc*	*scale*	*a*	*b*	*loc*	*scale*
GMRES	−5.86e−01	3.35	3.97e−04	1.07e−21	-7.40e−01	3.21	8.06e−04	1.86e−23
PGMRES	2.84	6.74	3.10e−04	2.26e−19	−6.18e−02	2.42	2.21e−03	6.47e−19
ex48								
GMRES	6.74e−01	2.09	2.22e−02	2.24e−24	6.69e−01	2.10	1.71e−03	9.23e−26
PGMRES	−6.02e−01	3.34	4.07e−04	3.05e−23	1.24	1.84	1.36e−02	1.66e−18

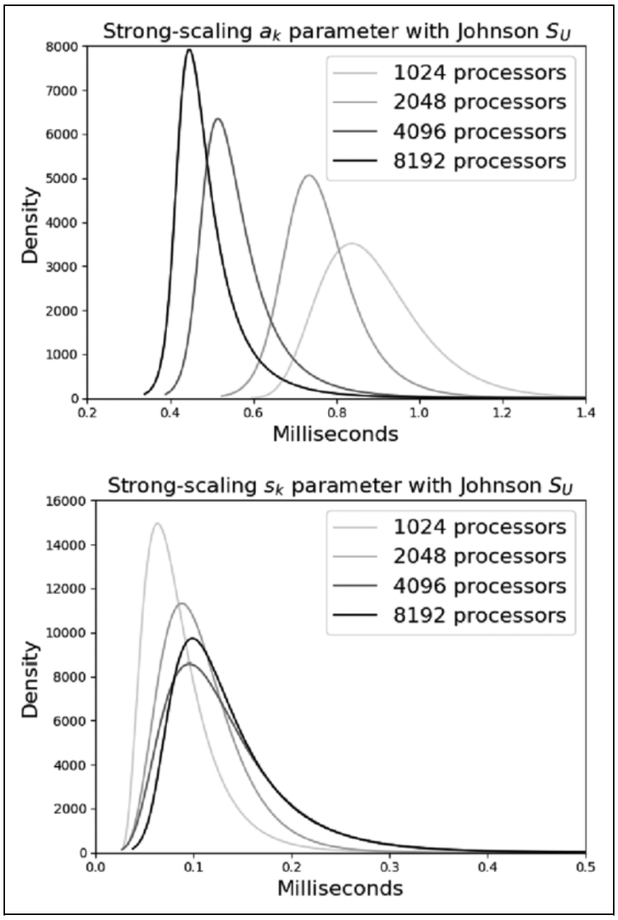

Figure 8. GMRES ex48 strong-scaling results on $P = 1024 - 8192$ processors.

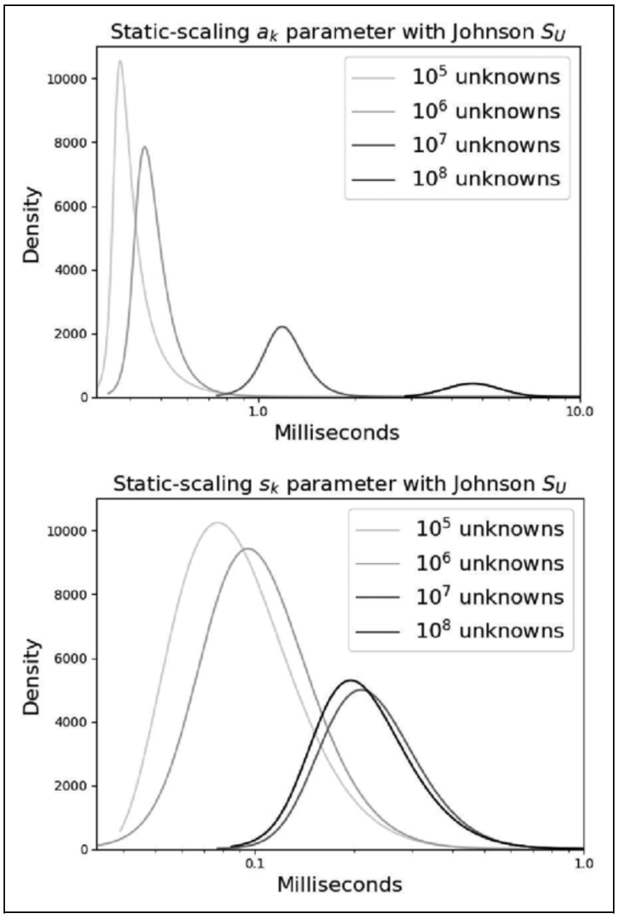

Figure 9. GMRES ex48 static-scaling results for $10^5 - 10^8$ unknowns.

factors, such as algorithm and matrix pattern, can influence performance. In this section, we expand our experiments to study other factors including problem size and computing platform and test our non-stationary model in more scenarios.

7.1 Scaling experiments

Our experiments so far have been performed on 8192 processors with 10^6 unknowns. To see how changing processor count and problem size affect the Krylov method performance and underlying uniform parameters, we run strong- and static-scaling experiments. Figures 8 and 9 show the Johnson S_U distributions for the uniform parameters a_k and s_k for strong- and static-scaling experiments on Theta.

With strong-scaling experiments, we see how varying the processor count affects Krylov method performance for a fixed problem. We repeat runs of ex48 with 10^6 unknowns on $P = 1024 - 8192$ processors. We see that, for a fixed number of unknowns, as we decrease the number of processors the Johnson S_U distribution shifts to the right and spreads out for the uniform a_k parameter. In this case, each processor does more work and is more likely to experience a detour. Similarly, we perform static-scaling experiments. In static-scaling, the problem size (or number of unknowns) is varied on a fixed number of processors. This allows us to see the effect of computation without changing the communication pattern. We keep the number of processors fixed at $P = 8192$ and vary the number of unknowns, which exaggerates what we see for strong-scaling as shown in Figure 9.

7.2 BiCGSTAB

We solve PETSc ex23, the one-dimensional Laplacian problem, using the Stabilized Biconjugate Gradient method (BiCGSTAB) (van der Vorst, 1992) and a pipelined version (PIPEBiCGSTAB) (Cools and Vanroose, 2017) with 5000 linear iterations and 10^6 unknowns on Theta. We choose these methods because their communication and performance characteristics differ somewhat from GMRES, as the methods are based on a short-term recurrence relation and do not require orthogonalizing against a growing set of basis vectors; BiCGSTAB and variants also require two matrix–vector multiplications per iterations, instead of one.

Many aspects of the BiCGSTAB and PIPEBiCGSTAB computations are consistent with GMRES and PGMRES, such as non-stationary iterates and nearly constant BiCG-STAB runtimes on a given node of Theta. Figure 10 shows uniform parameter a_k and s_k histograms with GMRES and PGMRES for comparison. It appears that BiCGSTAB and PIPEBiCGSTAB distributions can be modeled by the Johnson S_U distribution family. This suggests that our performance model is applicable to Krylov methods outside of GMRES methods. GMRES and PGMRES outperform BiCGSTAB and PIPEBiCGSTAB and have quicker minimum iteration times (Johnson S_U distributions shifted to the left for uniform parameter a_k), but runtimes are tightly grouped. Performance results for BiCGSTAB and PIPEBiCGSTAB using our non-stationary stochastic models are in shown Table 4.

7.3 Piz Daint

We repeat experiments on the Cray XC40 Piz Daint supercomputer[3] at the Swiss National Computing Center. Piz Daint and Theta both employ a Aries Dragonfly interconnect but Piz Daint contains Intel Xeon E5 Haswell processors (Hammarlund et al., 2014) on compute nodes (which provide more performance per CPU core compared to KNL). Therefore, the differences we see are likely due to different processors or the shared use of a communication network. Figure 11 shows runs of ex48 with GMRES and PGMRES on 8192 processors and 10^6 unknowns. In a given iteration, the quickest processor can be much faster on Piz Daint than Theta, but both uniform parameters contain much more variation. Non-stationary performance model results from Piz Daint are in shown Table 4.

7.4 Mira

We repeat ex48 experiments on Mira, an IBM Blue Gene/Q[4] (Kumaran, 2016) at Argonne's ALCF. We expect a

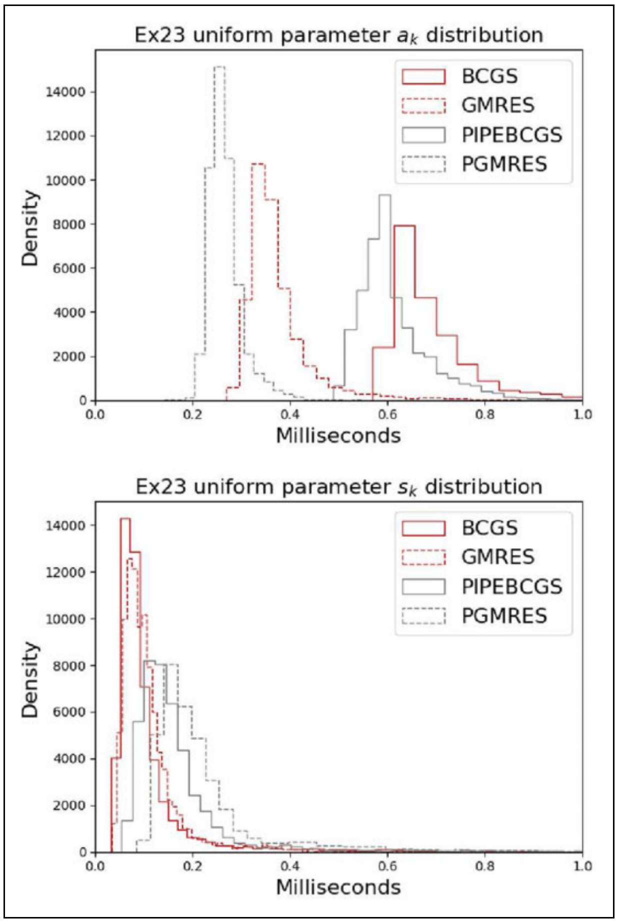

Figure 10. BiCGSTAB and PIPEBiCGSTAB uniform parameters compared to GMRES and PGMRES.

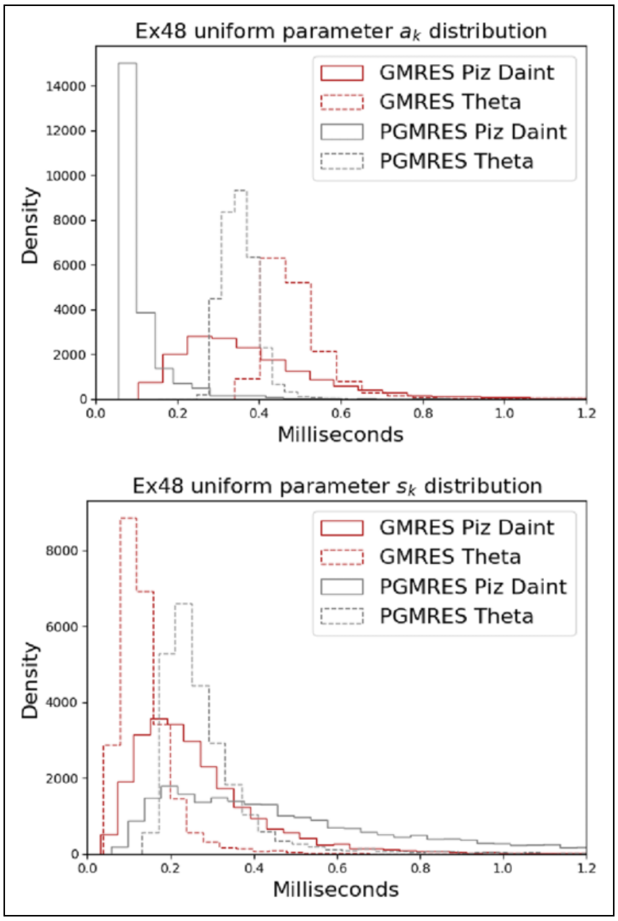

Figure 11. GMRES and PGMRES on Theta and Piz Daint.

Table 4. Non-stationary performance model results for ex23 and ex48 for extended experiments.

	Theta			Piz Daint		Mira	
ex23	**BiCGSTAB**	**PIPEBiCGSTAB**	**ex48**	**GMRES**	**PGMRES**	**GMRES**	**PGMRES**
KSPSolve	3.957	3.53	KSPSolve	3.029	2.642	3.026	2.804
$\hat{E}_{\mathrm{Unif}}[T]$	4.227	3.512	$\hat{E}_{\mathrm{Unif}}[T]$	3.541	2.473	3.189	2.455

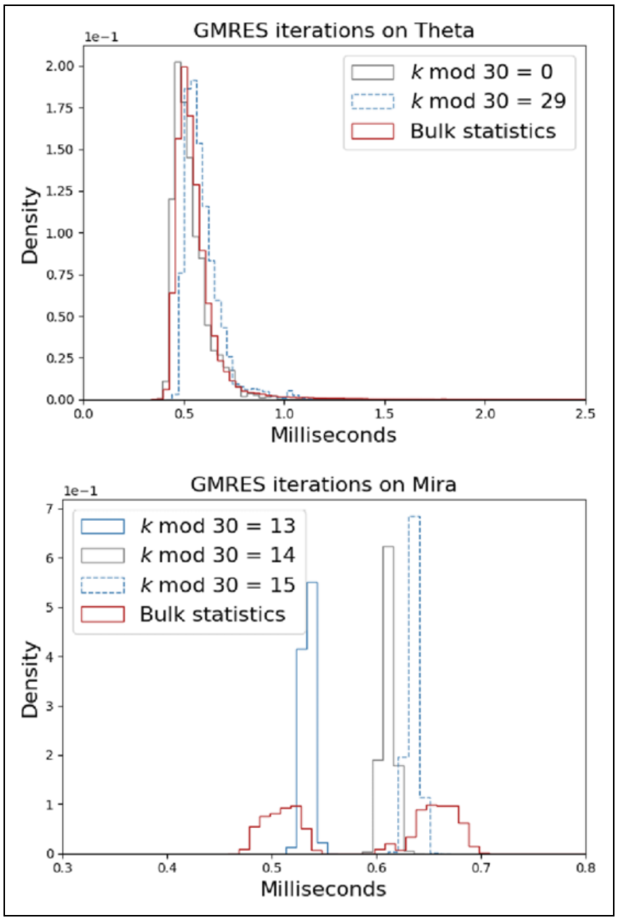

Figure 12. GMRES ex48 iteration density and Theta and Mira.

marked performance difference on Mira since nodes on Mira are connected by a 5D torus network with special hardware support for collective communication. Conversely, nodes on Theta are connected by a Cray Aries Network where neighboring workloads share network routes, increasing performance variability (Groves et al., 2017). Each Blue Gene/Q node also contains a redundant processor to service operating system interrupts, similar to Cray's core specialization feature where one core per node is dedicated to handling OS operations, which reduces core variability (Chunduri et al., 2017).

Figure 12 shows histograms of iterations from difference places in the GMRES restart cycle as well as a bulk histogram for all iterations from all processors, which can be thought of as the average performance. Each curve has been normalized to represent a distribution. GMRES iterations on Theta at different places in the cycle appear similar to the average. They are noisy enough that our simulation provided reasonable results even though we drew uniform parameters a_k and s_k from the same Johnson $\mathrm{S_U}$ distributions regardless of k in the GMRES cycle. With Mira's quiet network, the GMRES iterates are much more distinct suggesting that a refined model that accounts for the amount of work done per iteration would be needed for a priori estimates. Furthermore, we see a jump in the time between iterations $k \bmod 30 = 13$ and $k \bmod 30 = 14$, which is mostly likely due to L1 data cache effects as the required storage for basis vectors increases throughout a GMRES cycle. Similar results were found for PGMRES and performance results are in Table 4.

8 Conclusion and future work

In this work we gather fine-grained iteration data from runs of various Krylov subspace methods across multiple HPC platforms and for different computations. We examine the iterates over time and find that they are non-stationary. That is, the distribution of runtimes shifts with time even after accounting for the growing Krylov subspace. We see significant inter-node variability as well as some intra-node variability, possibly from reduction operations, synchronizations, or operating system interrupts, hindering the performance of traditional Krylov methods. Inhomogeneity from communication patterns is another source of variation between processor performance.

Using these insights, we develop a non-stationary, stochastic performance model that accurately reflects the performance of standard Krylov methods and pipelined algorithms across different algorithms and a variety of HPC architectures. Furthermore, this model suggests a way to make predictive performance estimates.

Future work should conduct experiments on a more comprehensive set of problems, gathering a wider range of runtimes for testing. We also plan to measure different components of Krylov iterations such as communication phases with orthogonalizations and make use of hardware counters. Also in the future, nondeterministic performance models can be derived for other parallel algorithms where unpredictable system interference could impact performance. We also plan on testing our model with heavier preconditioners, deeper pipelines, and more complex architectures. Since we made no assumptions about the hardware in our performance models, we could deploy the same experiments on, for example, a heterogenous machine with GPUs and analyze the data in a similar way.

Accurate performance models can assist in algorithmic choices and parameter tuning such as pipeline depth. Insights from performance models can also be used to guide the development of new algorithms, particularly those we expect to be running in less predictable computing environments, such as heavily loaded machines or loosely coupled networks used in cloud computing or those with shared resources.

Authors' note

reproduce, prepare derivative works, distribute copies to the public, and perform publicly and display publicly, by or on behalf of the Government. The Department of Energy will provide public access to these results of federally sponsored research in accordance with the DOE Public Access Plan. `http://energy.gov/downloads/doe-public-accessplan`. Hannah Morgan is now affiliated with University of Chicago.

Acknowledgments

Thanks to everyone who offered insight and advice, including Todd Munson, Barry Smith, Ivana Marincic, Vivak Patel, Karl Rupp, and Oana Marin.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The authors were supported by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research under Contract DE-AC02-06CH11357. This research used resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357. The authors were partially supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration. P.S. acknowledges financial support from the Swiss University Conference and the Swiss Council of Federal Institutes of Technology through the Platform for Advanced Scientific Computing (PASC) program.

ORCID iD

Hannah Morgan https://orcid.org/0000-0001-8830-6861

Notes

1. /src/ksp/ksp/examples/tutorials/ex23.c in PETSc 3.10.
2. src/snes/examples/tutorials/ex48.c in PETSc 3.10.
3. hybrid partition, November 2018, PETSc 3.10, Cray-MPICH 7.7.2.
4. November 2018, PETSc 3.10, MPICH2 1.5.

References

Agarwal S, Garg R and Vishnoi NK (2005) The impact of noise on the scaling of collectives: a theoretical approach. In: *International Conference on High-Performance Computing*, Goa, India, 18-21 October 2005, Berlin, Germany: Springer, pp. 280–289.

Alverson B, Froese E, Kaplan L, et al. (2012) Cray xc series network. *Cray Inc., White Paper WP-Aries01-1112*.

Balay S, Abhyankar S, Adams MF, et al. (2018a) PETSc users manual. Technical Report ANL-95/11—Revision 3.10, Argonne National Laboratory. Available at: http://www.mcs.anl.gov/petsc. (accessed 1 October 2019)

Balay S, Abhyankar S, Adams MF, et al. (2018b) PETSc Web page. Available at: http://www.mcs.anl.gov/petsc (accessed 1 October 2019).

Beckman P, Iskra K, Yoshii K, et al. (2006) The influence of operating systems on the performance of collective operations at extreme scale. In: *The International Conference on Cluster Computing*, Barcelona, Spain, 25–28 September 2006, pp. 1–12. New York, NY, USA: IEEE.

Bertsimas D, Natarajan K and Teo CP (2006) Tight bounds on expected order statistics. *Probability in the Engineering and Informational Sciences* 20(4): 667–686.

Chronopoulos AT and Gear CW (1989) *s*-Step iterative methods for symmetric linear systems. *Journal of Computational and Applied Mathematics* 25: 153–168.

Chunduri S, Harms K, Parker S, et al. (2017) Run-to-run variability on Xeon Phi based Cray XC systems. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Denver, CO, USA, 12-17 November 2017. New York, NY, USA: ACM, p. 52.

Cools S and Vanroose W (2017) The communication-hiding pipelined BiCGStab method for the parallel solution of large unsymmetric linear systems. *Parallel Computing* 65: 1–20.

Cramér H (2016) *Mathematical Methods of Statistics*, Vol. 9. Princeton: Princeton University Press.

David HA and Nagaraja HN (2004) Order statistics. *Encyclopedia of Statistical Sciences* 9: 5829–5838.

Ferreira KB, Bridges P and Brightwell R (2008) Characterizing application sensitivity to OS interference using kernel-level noise injection. In: *Proceedings of the Conference on Supercomputing*, Piscataway, NJ: IEEE/ACM. ISBN 978-1-4244-2835-9, pp. 19:1–19:12.

Ghysels P and Vanroose W (2014) Hiding global synchronization latency in the preconditioned conjugate gradient algorithm. *Parallel Computing* 40(7): 224–238.

Ghysels P, Ashby TJ, Meerbergen K, et al. (2013) Hiding global communication latency in the GMRES algorithm on massively parallel machines. *SIAM Journal on Scientific Computing* 35(1): C48–C71.

Gropp W and Lusk EL (1999) Reproducible measurements of MPI performance characteristics. In: *Proceedings of the Sixth European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface*, Barcelona, Spain, 26-29 September 1999, pp. 11–18 DOI:10.1007/3-540-48158-3_2.

Groves T, Gu Y and Wright NJ (2017) Understanding performance variability on the Aries dragonfly network. In: *International Conference on Cluster Computing*, Honolulu, HI, USA, 5–8 September 2017. New York, NY, USA: IEEE.

Hammarlund P, Martinez AJ, Bajwa AA, et al. (2014) Haswell: the fourth-generation Intel core processor. *IEEE Micro* 34(2): 6–20.

Hoefler T, Lumsdaine A and Rehm W (2007) Implementation and performance analysis of non-blocking collective operations for MPI. In: *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, Reno, NV, USA, 10-16 November 2007. New York, NY, USA: IEEE/ACM.

Hoefler T, Schneider T and Lumsdaine A (2010) Characterizing the influence of system noise on large-scale applications by simulation. In: *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. New Orleans, LA, USA, 13-19 November 2010. New York, NY, USA: IEEE/ACM, pp. 1–11.

Jacques T, Nicolas L and Vollaire C (1999) Electromagnetic scattering with the boundary integral method on MIMD systems. In: Yang T (ed) *Parallel Numerical Computation with Applications. The Springer International Series in Engineering and Computer Science*, Vol. 515. Boston, MA: Springer, pp. 215–230.

Kumaran K (2016) Introduction to Mira. In: *Code for Q Workshop*. Available at: https://cug.org/proceedings/cug2017_proceedings/includes/files/pap113s2-file1.pdf

Morgan H, Knepley MG, Sanan P, et al. (2016) A stochastic performance model for pipelined Krylov methods. *Concurrency and Computation* 28(18): 4532–4542.

Parker S, Morozov V, Chunduri S, et al. (2017) Early evaluation of the Cray XC40 Xeon Phi system Theta at Argonne. In: Cray User Group Proceedings. Available at: https://cug.org/proceedings/cug2017_proceedings/includes/files/pap113s2-file1.pdf

Pattyn F, Perichon L, Aschwanden A, et al. (2008) Benchmark experiments for higher-order and full Stokes ice sheet models (ISMIP-HOM). *The Cryosphere Discussions* 2(1): 111–151.

Saad Y (1996) *Iterative Methods for Sparse Linear Systems*. Boston, MA: PWS Publishing Company.

Seelam S, Fong L, Tantawi A, et al. (2010) Extreme scale computing: Modeling the impact of system noise in multicore clustered systems. In: *International Symposium on Parallel & Distributed Processing*, Atlanta, GA, 19-23 April 2010. New York, NY, USA: IEEE, pp. 1–12.

Sodani A (2015) Knights landing (KNL): 2nd generation Intel Xeon Phi processor. In: *Hot Chips 27 Symposium*, Cupertino, CA, 23-25 August 2015. New York, NY, USA: IEEE, pp. 1–24.

Strzodka R and Göddeke D (2006) Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components. In: *Proceedings of the Symposium on Field-Programmable Custom Computing Machines*, Napa, CA, 24-26 April 2006. New York, NY, USA: IEEE, pp. 259–270.

Sturler ED and van der Vorst HA (1995) Reducing the effect of global communication in GMRES(m) and CG on parallel distributed memory computers. *Applied Numerical Mathematics* 18: 441–459.

van der Vorst H (2003) *Iterative Krylov methods for Large Linear Systems*. Cambridge: Cambridge University Press. ISBN 9780521818285.

van der Vorst HA (1992) Bi-CGSTAB: A fast and smoothly converging variant of Bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific Computing* 13(2): 631–644.

Author biographies

Hannah Morgan is a postdoctoral researcher in the Mathematics and Computer Science Divison at Argonne National Laboratory. Her PhD work involved efficiently simulating fluid models as well as performance modeling. Currently, she is working on the performance of the PETSc scientific computing library on hybrid CPU/GPU architectures as part of the Exascale Computing Project.

Patrick Sanan is a postdoctoral researcher in the Institute of Geophysics at ETH Zurich. His interests include applied mathematics, scalable solvers, scientific software, and computational Earth sciences.

Matthew Knepley is an associate professor in the Department of Computer Science and Engineering at the University at Buffalo. His research focuses on scientific computation, including fast methods, parallel computing, software development, numerical analysis, and multicore architectures. He is an author of the widely used PETSc library for scientific computing from Argonne National Laboratory.

Richard Tran Mills is a principal computational engineer in the Mathematics and Computer Science Division at Argonne National Laboratory. His research has spanned high-performance scientific computing, geospatiotemporal data mining and machine learning, computational hydrology, and climate change science. He is one of the original developers of PFLOTRAN, an open-source code for massively parallel simulation of hydrologic flow and reactive transport problems, and is a core developer of PETSc, the Portable, Extensible Toolkit for Scientific Computation. He earned his Ph.D. in Computer Science in 2004 at the College of William and Mary, where he was a Department of Energy Computational Science Graduate Fellow. Prior to that, he studied geology and physics at the University of Tennessee, Knoxville as a Chancellor's Scholar.