A Scalable Framework for Solving Fractional Diffusion Equations

MAX CARLSON, University of Utah
ROBERT M. KIRBY, University of Utah
HARI SUNDAR, University of Utah

The study of fractional order differential operators (involving non-integer derivative terms) is receiving renewed attention in many scientific fields from photonics to speech modeling. While numerous scalable codes exist for solving integer-order partial differential equations (PDEs), the same is not true for fractional order PDEs. Therefore, there is a need for highly scalable numerical methods and codes for solving fractional order PDEs on complex geometries. The key challenge is that most approaches for fractional PDEs have at least quadratic complexity in both storage and compute, and are challenging to scale. We present a scalable framework for solving fractional diffusion equations using the method of eigenfunction expansion. This includes a scalable parallel algorithm to efficiently compute the full set of eigenvalues and eigenvectors for a discretized Laplace eigenvalue problem and apply them to construct approximate solutions to fractional order model problems. We demonstrate the efficacy of our methods by performing strong and weak scalability tests using complex geometries on TACC's Frontera compute cluster. We also show that our approach compares favorably against existing dense and sparse solvers. In our largest solve, we estimated half a million eigenpairs using 28,672 cores.

CCS Concepts: • **Computing methodologies → Distributed algorithms**; **Parallel algorithms**; • **Mathematics of computing → Mathematical software performance**; **Solvers**.

ACM Reference Format:
Max Carlson, Robert M. Kirby, and Hari Sundar. 2020. A Scalable Framework for Solving Fractional Diffusion Equations. In *2020 International Conference on Supercomputing (ICS '20), June 29-July 2, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3392717.3392769

1 INTRODUCTION

Fractional partial differential equations (PDEs) are differential equations involving real number powers—as opposed to the more traditional integer powers—of the differentiation operator. Fractional PDEs have been used to model a wide range of problems such as propagation of acoustical waves in biological tissue [Holm and Näsholm 2011], photonics [Chen et al. 2019], and speech modeling[Assaleh and Ahmad 2007]. While several scalable methods and codes are available for solving integer-order PDEs, scalable computational methods and codes are not available for fractional PDEs. This work aims to fill this gap by presenting the first large-scale scalable framework for solving fractional PDEs on complex geometries. Specifically, we explore the method of eigenfunction expansion as a method for discretizing the fractional Laplacian operator for use in solving space-fractional elliptic and parabolic PDEs. This formulation of the fractional Laplacian requires computing full set of eigenvalues and eigenvectors for the discretized Laplacian (∇^2, non-fractional).

Due to the non-local nature of fractional differential operators, it is necessary to compute the full spectrum instead of only a small subset, as is common for several other applications requiring scalable eigensolvers. Computing the full spectrum for a discretized operator is a task usually done by dense, direct solvers. However, the scalability for direct eigensolvers is poor in the distributed memory setting. Additionally, the performance of direct solvers on complex geometry was dramatically slower than on simple domains.

Alternatively, sparse iterative solvers–such as SLEPc [Hernandez et al. 2005]–are highly efficient at producing small subsets of eigenpairs but convergence slows down significantly while estimating larger sets of eigenpairs. In addition, they suffer from issues related to partitioning work that hurts overall scalability. Our work focuses on solving these scalability issues to enable the use of iterative eigensolver techniques for estimating the full spectrum of large problems.

In order to compute the full set of eigenpairs using sparse solvers, a method called spectrum slicing can be employed. This method involves slicing up the spectrum into sub-intervals that can be solved independently for all eigenpairs in the sub-interval. This addresses the degrading convergence issues related to estimating large sets of eigenpairs, as well as provides an added level of parallelism. However, since the distribution of eigenvalues is not known, and can be strongly skewed, determining the appropriate intervals to partition the spectrum is not trivial.

Without a proper partitioning algorithm, any scalability to be gained from spectrum slicing can easily be outweighed by the time it takes to set up the problem.

In this work, we present a method for partitioning the spectrum into sub-intervals such that each sub-interval contains equal number of eigenpairs and thus takes roughly the same amount of time to solve. This allows each independent sub-problem to be solved in parallel with minimal load imbalance and results in a highly scalable method for constructing the eigenbasis necessary for solving fractional PDEs. The key property of our partitioning scheme is that it has time complexity that is constant with respect to the number of processors. This property is crucial for achieving scalability using a spectrum slicing approach. Using our scheme, we demonstrate that we can achieve super-linear parallel efficiency at extreme scale and can compute up to half a million eigenpairs in under 10 minutes which would not be achievable using dense direct solvers or sparse iterative solvers like SLEPc[1]. Our approach outperforms and scales better than the ScaLAPACK [Blackford et al. 1997] dense, direct

ICS '20, June 29-July 2, 2020, Barcelona, Spain
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7983-0/20/06…$15.00
https://doi.org/10.1145/3392717.3392769

[1]the default SLEPc solver. This work is built on top of SLEPc.

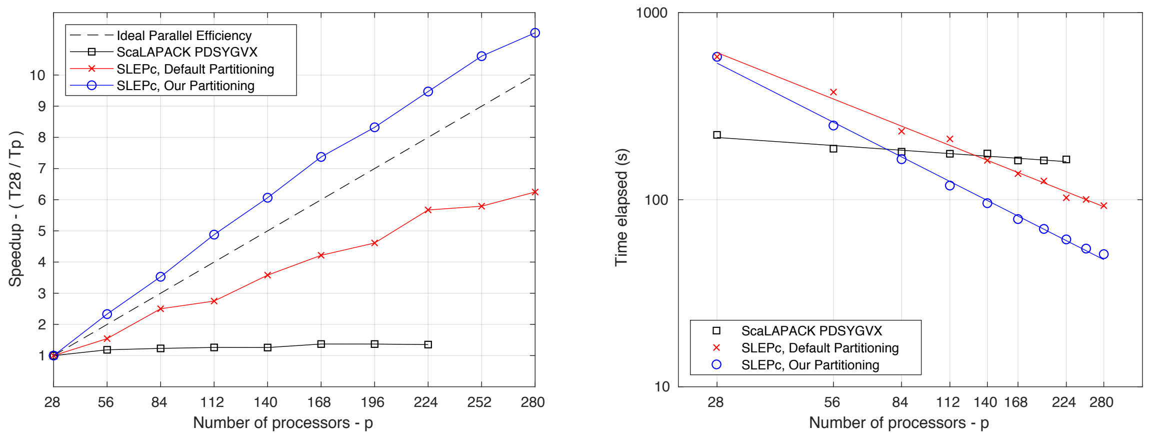

Fig. 1. **(left)** Parallel efficiency comparison of ScaLAPACK's dense eigenvalue solver (PDSYGVX) plotted against the parallel efficiency of SLEPC's sparse solver using default partitioning and our partitioning algorithm. **(right)** Strong scaling comparison of ScaLAPACK's dense eigenvalue solver (PDSYGVX) plotted against the strong scaling of SLEPC's sparse solver using default partitioning and our partitioning algorithm. This experiment was run on a 25 by 25 by 25 cube mesh. Note: The baseline for computing speedup was using the full set of 28 processors available for a node. As a result, the scalability for ScaLAPACK appears flat due to most of the speedup happening between 1 and 28 processors and falling off once reaching the distributed setting.

eigenvalue solver (PDSYGVX) as well as SLEPc's sparse iterative solvers. In figure 1, we show the parallel efficiency and runtime of ScaLAPACK (dense, direct), vanilla SLEPc (sparse, iterative), and SLEPc equipped with our partitioning scheme. We also demonstrate that this numerical scheme has exponentially convergent accuracy when solving fractional diffusion or Poisson equations.

Contributions: To the best of our knowledge, this is the first large-scale solver for fractional diffusion equation. Our solver supports complex 2D and 3D domains. A central contribution has been the development of a scalable spectrum slicing algorithm that improves the scalability and load-balance of iterative eigensolvers when estimating large number of eigenpairs. This contribution should be have an impact to a wider range of problems beyond fractional PDEs. We also demonstrate the excellent speedup—superlinear in many cases—as well as comparisons with state-of-the-art direct and iterative eigensolver. Our code is built on top of the SLEPc/PETSc libraries and is open source and available under the MIT license [2]. While the core fractional component is discretization agnostic, we have also integrated with the open source *hp*-finite element library `nektar++`, to enable domain scientists to solve fractional Laplacian problems on complex 2D and 3D domains.

The rest of the paper is organized as follows. In §2 we present the mathematical details and background for non-local operators and how the properties of such operators motivated the need for our algorithm. We then present the method and implementation details in §3. Then we demonstrate the scalability and performance of our approach in §4 and the accuracy of the method in §5.

[2]`github` https://github.com/paralab/Nektarpp_EigenMM

2 NON-LOCAL OPERATORS

While it isn't necessary to understand the specific details of fractional differential operators to understand the eigenvalue partitioning algorithm that we are presenting, it does provide context for the numerical and computational challenges that are unique to these operators. In this section, we will give some background on these specific challenges and how they have shaped our approach.

Fractional differential operators are characterized by their non-local property. For instance, time differential equations with integer order operators describe processes that are based on the assumption that the rate of change of some value is completely independent on the state of the process in the past. This is a perfectly fine assumption for a lot of naturally occurring processes but it does not hold in general.

One example of how expanding models to include non-local fractional operators furthered the understanding of a process comes from diffusion. [Henry et al. 2010] [Oliveira et al. 2019] One of the key features of diffusion that comes from the integer order model is that the mean squared displacement scales proportionally with time. There have been experiments that show there are processes that look very much like diffusion but do not satisfy this condition. This type of diffusion process is known as anomalous diffusion and can be modeled using fractional time or space differential operators depending on the specific regime. For example, superdiffusion (one of the regimes of anomalous diffusion) can be modeled using the fractional Laplacian operator and is the type of process we are interested in.

For this paper, we are specifically interested in the fractional Laplacian operator [Lischke et al. 2018]. It shows up in many models ranging from finance [Kumar et al. 2014] [Levendorskii 2004], biology [Cusimano et al. 2015] [Magin 2010] [Cusimano et al. 2013], structural mechanics [Tarasov and Aifantis 2018] [Evgrafov and Bellido 2018], and quantum mechanics [Laskin 2002] [Guerrero and

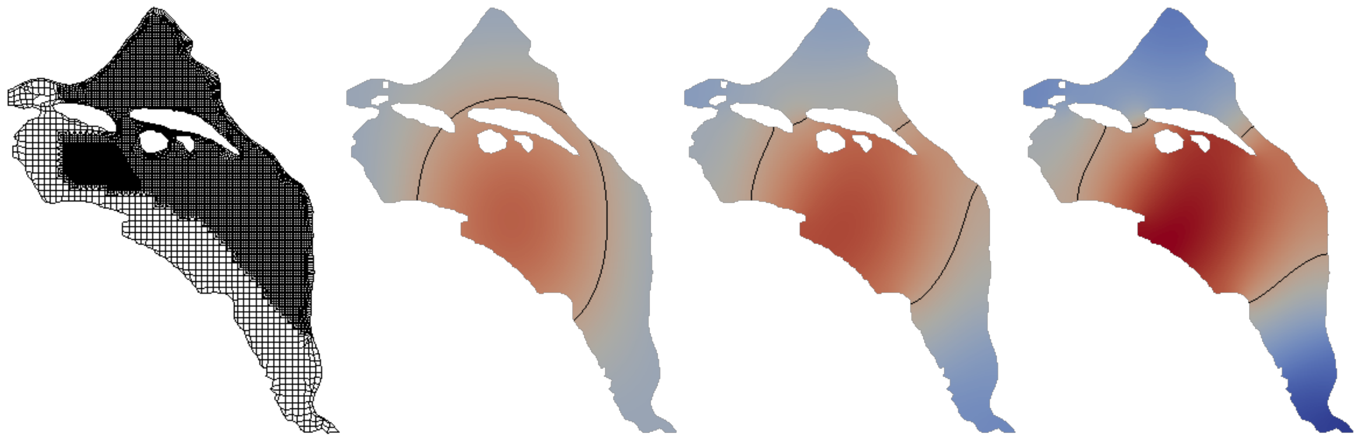

Fig. 2. Hanford site mesh and three example solutions for the fractional Poisson problem (6). Solution plots are, from left to right, for $\alpha = 0, 1, 2$. Black lines in the solution plots indicate the contour where $u(x) = 0$. The fractional Laplacian with degree α is equivalent to the standard Laplacian when $\alpha = 2$, and is equivalent to the identity when $\alpha = 0$, that is $(-\Delta)^0 u = u = f$.

Moreles 2015]. Most of these models are interested only in the 1D version of the fractional Laplacian or are limited to simple domains in higher dimensions. This may be partially due to the lack of widely available solvers for fractional PDEs on very large complex domains.

With this research happening in fractional PDEs, it is necessary to have scalable solvers for complex geometry so that scientists can verify and replicate results numerically in a reasonable amount of time. We have developed one such solver and the following sections of this paper will go into detail on how our method works and the paralellization strategies employed to ensure we are taking full advantage of available computing resources.

2.1 Spectral Fractional Laplacian

For this paper, we will use the spectral definition of the fractional Laplacian [Lischke et al. 2018] specifically for the case of homogeneous boundary conditions (Dirichlet or Neumann). This spectral definition is constructed by first considering the infinite, discrete set of eigenvalues and eigenfunctions of the integer order Laplace eigenvalue problem (1) with appropriate homogeneous boundary conditions. It is possible to extend this method to nonhomogeneous boundary conditions by using harmonic lifting but this is beyond the scope of this paper as it does not change the underlying principles of the method. Harmonic lifting can be computed using integer order solvers for the Laplace equation and this does not impact how the fractional Laplacian operator is constructed.

$$\begin{aligned} -\Delta\psi_k &= \lambda_k\psi_k \\ \mathcal{B}(\psi_k)|_{\partial\Omega} &= 0. \end{aligned} \tag{1}$$

With the eigenpairs of (1), the spectral fractional Laplacian of a function u is defined as

$$(-\Delta)^{\alpha/2}u = \sum_{k=1}^{\infty} \lambda_k^{\alpha/2}\langle u, \psi_k\rangle\psi_k \tag{2}$$

where $\langle\cdot,\cdot\rangle$ is the inner product for $L_2(\Omega)$. Refer to [Lischke et al. 2018] for a more elaborate treatment of this and more definitions for the fractional Laplacian operator.

The first model problem we will consider is the space-fractional diffusion equation (3) with homogeneous boundary conditions and some arbitrary initial conditions:

$$\begin{aligned} \partial_t u(x,t) &= -\mu(-\Delta)^{\alpha/2}u \\ u(x,t)|_{\partial\Omega} &= 0 \\ u(x,0) &= u_0(x). \end{aligned} \tag{3}$$

The spectral definition of the fractional Laplacian can be plugged directly into the diffusion equation (3) and using linearity we obtain

$$\sum_{k=1}^{\infty}[\partial_t\hat{u}_k(t) + \mu\lambda_k^{\alpha/2}\hat{u}_k(t)]\psi_k = 0 \tag{4}$$

This equation is satisfied when for all $k \in 1, 2, 3, \ldots$, each coefficient $\hat{u}_k(t)$ satisfies the ODE $\hat{u}_k' = -\mu\lambda_k^{\alpha/2}\hat{u}_k$. These ODEs have exact solutions $\hat{u}_k(t) = \hat{u}_k(0)e^{-\mu\lambda_k^{\alpha/2}t}$. Therefore, this fractional diffusion problem has the exact solution

$$u(x,t) = \sum_{k=1}^{\infty} e^{-\mu\lambda_k^{\alpha/2}t}\langle u_0, \psi_k\rangle\psi_k. \tag{5}$$

Similarly, we consider the steady state of the diffusion equation with a forcing term. This equation (6) is also known as the fractional Poisson equation with homogeneous boundary conditions:

$$\begin{aligned} (-\Delta)^{\alpha/2}u &= f \\ u|_{\partial\Omega} &= 0. \end{aligned} \tag{6}$$

If f can be adequately represented by a spectral series that satisfies our boundar conditions,

$$f = \sum_{k=1}^{\infty}\langle f, \psi_k\rangle\psi_k \tag{7}$$

then we can once again directly plug the spectral definition and this f into (6). Using the linearity of summations and orthogonality of the eigenfunctions, the coefficients for the unknown function u can be solved for directly to obtain

$$u = \sum_{k=1}^{\infty} \lambda_k^{-\alpha/2} \langle f, \psi_k \rangle \psi_k. \tag{8}$$

For both of these model problems, we have not needed to discuss a numerical scheme for computing solutions. We have simply shown that both problems have an extensive family of exact solutions that contain the types of solutions we are looking for. For domains like lines, squares, and cubes, this eigenfunction expansion definition is simply a more general form of the Fourier series solution.

This more general definition also allows us to extend this approach to complex geometries with known exact eigenfunctions like discs, spheres and cylinders. For instance, the eigenfunctions for a sphere are the spherical harmonics and can be evaluated using the fast spherical harmonic transform [Mohlenkamp 1999].

This also means we can use this method to solve problems on arbitrary complex geometry by solving the (integer order) Laplace eigenvalue problem for approximations to the appropriate eigenfunctions. By discretizing the domain and using the weak form of the Laplace eigenvalue problem we obtain

$$K\phi_k = \theta_k M \phi_k \tag{9}$$

where K is the stiffness matrix and M is the mass matrix such that

$$K_{ij} = \langle \nabla e_i, \nabla e_j \rangle \quad M_{ij} = \langle e_i, e_j \rangle \tag{10}$$

given basis functions e_i and e_j.

Using the solution to this eigenvalue problem, the approximate solution is simply

$$\begin{aligned} g &= \textstyle\sum_{k=1}^{N} \langle f, \phi_k \rangle \phi_k \\ v &= \textstyle\sum_{k=1}^{N} \theta_k^{-\alpha/2} \langle f, \phi_k \rangle \phi_k \end{aligned} \tag{11}$$

where g is the approximation of the forcing function using the computed eigenbasis and v is the approximate solution to the fractional Poisson problem (6).

This system (9) is just the typical finite element discretization [Boffi 2010] of the Laplace eigenvalue problem and therefore the resulting K and M have the usual sparsity that can be leveraged. The tradeoff then is instead of solving an $N \times N$ dense system (fractional FEM, finite differencing, etc), we solve for N eigenpairs of the sparse, symmetric system (9).

3 METHOD

To solve the fractional Poisson problem (6), the overall algorithm then can be seen in Algorithm 1. For a given input geometry, the eigenvalue problem is solved for all eigenpairs to construct the fractional operator. Then to solve the fractional Poisson problem for a given forcing function f, the approximate solution can be computed using only a few matrix-vector multiplications.

Solving a dense $N \times N$ system is an $O(N^3)$ operation. Alternatively, solving for the full set of eigenpairs for a given mesh can be done in $O(N^2)$ time and space by exploiting the sparsity of the integer order eigenvalue problem. For a given complex geometry, the full set of eigenpairs need to be only computed once and can be reused to solve multiple problems. Solving a single problem only requires applying the dense fractional operator which can be done in $O(N^2)$ time and space. Therefore this approach is a much more cost-effective strategy for handling non-local fractional problems.

Algorithm 1 Solver Pipeline: Fractional Poisson

```
1: Form standard (non-fractional) stiffness/mass matrices: K, M
2: Solve for approximate eigenpairs: $\Theta, \Phi$
3: for each forcing function f do
4:     Compute FEM expansion coefficients of f: $\bar{f}$
5:     Estimate spectral coefficients: $\hat{f} = \bar{f}^T M \Phi$
6:     Compute solution coefficients: $\hat{u} = \Phi \Theta^{-\alpha/2} \hat{f}$
7:     Evaluate FEM expansion to get approximate solution
8: end for
```

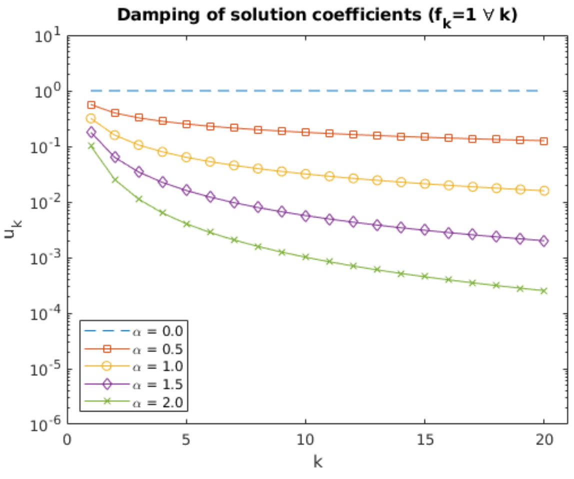

Fig. 3. Applying the fractional Laplacian can be viewed as a damping of the spectral coefficients. This example is what the damped coefficients look like for various α for the first 20 1D eigenvalues.

To solve the Laplace eigenvalue problem, we need a discrete system. Any finite element discretization will suffice as long as we can assemble the stiffness and mass matrices for the non-fractional Laplace eigenvalue problem. The main operation that dominates the runtime of this solver is then solving for all eigenpairs of this discrete system 9. Since we are solving for all N eigenpairs of a sparse system, the time and space complexity is at least $O(N^2)$. Therefore we need a parallelization strategy that reduces this amount of work as much as possible and takes full advantage of available hardware.

3.1 Partitioning

We now present our approach to partitioning the total set of eigenpairs into equally sized, contiguous groups of eigenpairs that can be solved for independently. The naive approach (and default SLEPc behavior) is to form P sub-intervals of equal length ρ/P but spectral density is not typically uniformly distributed. In figure 4, the cumulative eigenvalue counts can be seen for two large meshes. For the

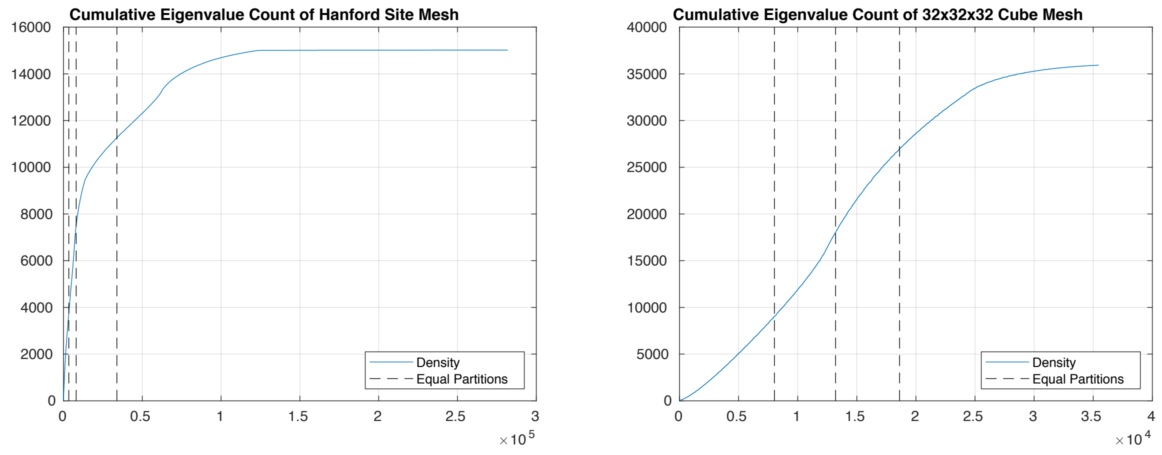

Fig. 4. The cumulative eigenvalue counts for the (**left**) Hanford site mesh (~15k DoFs) and a (**right**) 32x32x32 cube mesh (~32k DoFs). The dashed lines indicate the points at which the interval can be split into 4 sub-intervals with equal eigenvalue counts.

cube, the density is somewhat uniform and the point at which the interval could be split into two equal halves is near 40%. Alternatively, the Hanford site (see §4) mesh's density is very non-uniform and half of the eigenvalues reside in the first 3% of the spectrum and there are gaps with no eigenvalues. This type of non-uniform spectral density is much more typical of complex geometry and illustrates the need for our efficient and accurate partitioning.

In order to achieve this load-balanced partitioning, we need some way to count the number of eigenvalues in a sub-interval. For a given system, the spectral radius ρ can be approximated with a few applications of K and M. Since the system is symmetric, we know all possible eigenvalues are real and lie in the interval $[-\rho, \rho]$. For the standard Laplace problem, we also have positive semi-definiteness so this can be taken even further to say all eigenvalues are in $[0, \rho]$. Then, in order to achieve maximum parallel efficiency, we need to divide this interval into P independent sub-intervals with equal eigenvalue counts.

3.2 Solving for all eigenpairs in an interval

One family of algorithms for solving sparse linear eigenvalue problems are the Krylov subspace methods. The idea is to produce a tridiagonal matrix iteratively that has eigenvalues and eigenvectors that are approximately equal to the true eigenpairs. At each iteration, both the number of columns and rows of the tridiagonal matrix increases by one, which also means the number of approximate eigenvalues increases by one.

Krylov subspace methods compute a pre-determined number of eigenpairs close to a supplied target value. This is achieved by forming a shifted system $(K - aM)$ centered around the point a. The true eigenvalues of this shifted system close to 0 can be shifted back to get the true eigenvalues of the original system that are closest to a. With this, we can iteratively solve for the d closest eigenpairs to any given value a.

Solving for the d closest eigenvalues to a and the d closest eigenvalues to another point b can done completely indepedently of each other. If the total set of eigenvalues can be partitioned into groups of d eigenpairs, then we can use SLEPc's Krylov-Schur iterative method to compute each batch of eigenpairs completely indepedently. This forms the first level of parallelism for our approach and is typically known as spectrum slicing [Li et al. 2018a]. The challenge is then how to split the interval containing all solutions into load-balanced sub-problems.

3.3 Two-level parallelism and communication hierarchy

We utilize a two-level approach to parallelism by grouping MPI tasks into processor teams called evaluators. Each evaluator handles solving one sub-interval for N/P eigenpairs and the p MPI tasks within each evaluator compute the parallel Cholesky factorization for its sub-problem. By default, P is the number of nodes available and p is the number of processors available on a single node. However, the communication hierarchy is flexible and a user can set it so there are multiple evaluators per node or multiple nodes per evaluator.

It is important to note here that as the number of degrees of freedom N increases, the size of the system $(N \times N)$ is increasing *and* the total number of eigenpairs that we need to compute is increasing. The two layer parallelism is then designed to efficiently tackle both directions of growth. The number of evaluators P grows in order to decrease the number of eigenpairs (N) per sub-interval. The number of processors per evaluator p then grows to match the optimal number of threads to do the $(N \times N)$ sparse Cholesky factorization.

3.3.1 Eigenvalue counting. One approach for counting the number of eigenvalues in an interval is to utilize Sylvester's Law of Inertia. Specifically, we can construct the shifted system (centered at a) and compute its Cholesky factorization $A_a = K - aM = LDL^T$. The number of eigenvalues greater than a is equal to the number of positive entries in the diagonal matrix D. If we repeat this process on the shifted system $A_b = K - bM$ we get the total number of eigenvalues in the interval $[a, b]$.

Alternatively, a cheaper approximate technique for counting eigenvalues is Kernel Polynomial Filtering [Napoli et al. 2013]. This technique is used typically on extremely large matrices when the exact number of eigenvalues is not important. This method is quite a bit cheaper since it only requires matrix-vector multiplications and not a full Cholesky factorization. Unfortunately, to get even close to the accuracy needed for any partitioning algorithm would require so many matrix-vector multiplications that the Cholesky factorization would be far cheaper.

SLEPc has no methods for actually forming the independent sub-intervals and if the partitioning is not supplied, it will use evenly spaced sub-intervals which result in massive load imbalance even on simple domains. Therefore we had to develop our own partitioning scheme to get proper load balancing. Since we don't know exactly where all of the N eigenvalues are located, we use the exact eigenvalue counting technique as a building block for sampling the density of the spectrum.

3.3.2 *Tree partitioning.*

With the ability to count the number of eigenvalues greater than some point a, we could do a binary search to split the total interval into two sub-intervals of arbitrary size. Then for any given P, the interval could be split in a tree-like fashion (an example can be seen in Figure 5) to get evenly loaded sub-intervals. Unfortunately, this approach had a number of problems that dramatically hurt scalability.

The total amount of work required to do the partitioning using this scheme is

$$\tau_P \approx \log_2(P) n_c T(k, N, p), \tag{12}$$

where P is the number of evaluators, p is the number of processors per evaluator, k is some measure of the sparsity of the system, n_c is the number of iterations the binary search takes to get a good split, and T is the time it takes to do the parallel Cholesky factorization of the shifted system $K - aM$. This time $T(k, N, p)$ looks like $\frac{kN}{/} p$ for small values of p but the parallel efficiency hits diminishing returns as p grows.

Even with the fact that $log_2(P)$ grows slowly with P, the $n_c T(k, N, p)$ portion is expensive enough that multiplying it by some scaling factor is not going to have the scaling properties that we need. Additionally, during tree partitioning, many of the evaluators don't have any work to do which means utilization for this approach is very low.

3.3.3 *Iterative partitioning.*

Not only did we want to avoid scaling the partitioning time with the number of evaluators, but if possible, avoid doing any binary searches as well. Each binary search requires around 10 Cholesky factorizations in order to get a good split which adds up really fast. Additionally, all of those 10 steps have to be done by a single evaluator and cannot really be aided by any idle machines. Therefore we came up with a greedy iterative partitioning strategy that works as follows.

We can get an initial guess for the partitioning by splitting the total interval into P sub-intervals of equal length. This is actually the default behavior of SLEPc if no partitions are provided to a spectrum slicing method. This initial guess is results in very poor

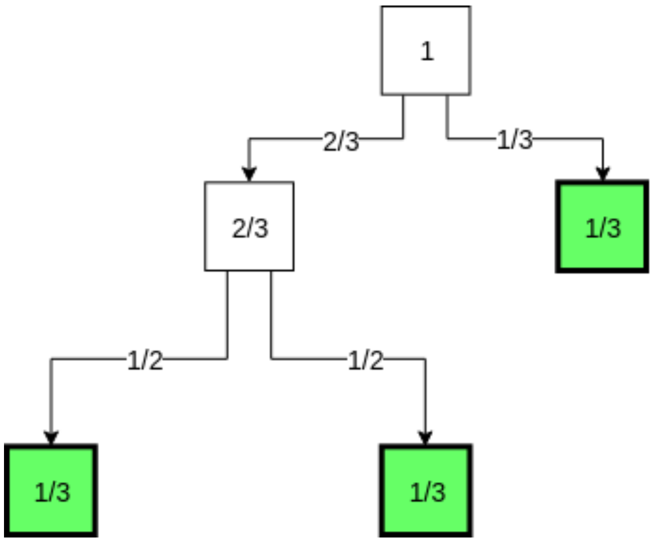

Fig. 5. Tree based partitioning. In order to split all eigenvalues into three equal-sized groups, first split the total interval into 2/3 and 1/3 sized groups and then split the 2/3 sized group in half. Each split involves a binary search style operation where each step requires a Cholesky factorization.

scalability even for simple geometries since eigenvalues are not evenly distributed through the spectrum.

With this initial guess, all but one evaluators each compute the number of eigenvalues greater than one of the splitting points in a single Cholesky factorization in parallel. With these values, the number of eigenvalues in each sub-interval can be computed and can be seen plotted in Figure 6. This histogram can be thought of as a piecewise constant approximation to the true density of states for the system. Then the root processor uses this approximation of the density to choose new splits that would give equally loaded sub-intervals if this was the true density and then this process repeats n_a times where n_a is some user supplied parameter.

This greedy iterative approach does not converge to the true optimal partitioning. Instead it gets very close to optimal and then enters a cycle. However, the best partitioning can be stored at each step and is reached usually within 3 or 4 iterations. This greedy global refinement only takes a handful (n_a has a default of 7) of Cholesky factorization and provides a partitioning that is much closer to the optimal than the naive initial guess.

This iterative approach can be further modified to get exact convergence by merging the eigenvalue counts at each step so that the piecewise constant approximation is increasing in resolution at each step nearby the optimal solution. A sketch of this modified approach can be seen in Algorithm 2. The tradeoff here is that for some extra work, the iterative partitioning will converge exactly. The greedy approach typically gets close enough to the ideal partitioning that it usually is not worth the extra work to partition exactly.

In certain rare cases, due to precision issues, the exact eigenvalue counting technique will give a count that is slightly incorrect. When this happens, the count for a sub-interval can go negative and two neighboring partitions will end up out of balance. For these cases, a

Algorithm 2 Adaptive Partitioning

```
1:  Pick equally spaced initial guess $x^{(0)}$
2:  Evaluator $i$ computes number of eigenvalues greater than $x_{i+1}^{(0)}$
    and stores the result in $R_i^A$
3:  for $k = 1 \ldots n_k$ do
4:      Refine $x^{(k-1)}$ to get $x^{(k)}$ using $R^A$
5:      Count sub-intervals and put result in $R^B$
6:      if $k < n_k$ then
7:          Merge $(x^{(k-1)}, R^A)$ and $(x^{(k)}, R^B)$
8:          Store result in $(x^{(k)}, R^A)$
9:      end if
10: end for
```

second optional local refinement stage can be employed. In order to do this in such a way that does not scale with P, for each iteration, half of the evaluators do a binary search on their pair of sub-intervals (if they are unbalanced beyond a given threshold) to balance the pair towards the optimal and then other half of evaluators balance the remaining pairs. This can then be done over n_b passes but typically one pass is sufficient.

With this strategy, the total time complexity for partitioning is

$$\tau_P \approx (n_a + 2n_b n_c)T(k, N, p) \tag{13}$$

where n_a is the number of global refinement steps (default 7), n_b is the number of local refinement passes (default 3), and n_c is the number of iterations required to get a good split by a binary search (default 10). This is exactly the kind of time complexity we are looking for. The number of evaluators P does not show up at all and the total number of Cholesky factorizations is completely determined by user-supplied constants. It should also be noted that this is a worst case complexity. In practice, the local refinement is rarely triggered and even then usually terminates in a single pass.

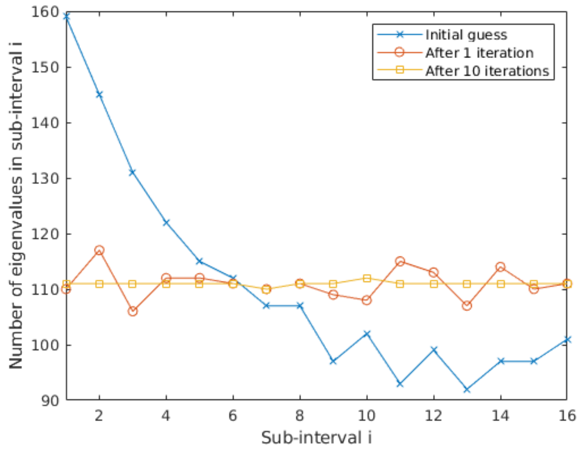

Fig. 6. Iterative partition refinement. The initial guess of equally space sub-intervals (default behavior in SLEPc) results in unbalanced partitions.

3.4 Post-processing

Finally, once the complete set of eigenpairs are computed, they need to be orthogonalized and normalized to ensure the eigenvectors are orthonormal. Fortunately, since the system is symmetric, the only eigenvectors that need to be made orthogonal to each other are those that share the same eigenvalue. Since any eigenvectors that share an eigenvalue will fall into the same sub-interval, the same evaluator that computed these vectors can do the orthogonalization completely independently from any other evaluator.

4 PERFORMANCE

The overall time complexity using our partitioning algorithm is then

$$\tau \approx \frac{N}{P}T(k, N, p) + (n_a + n_b n_c)T(k, N, p) + R \tag{14}$$

where the first term is the eigenvalue problem solve phase, the second term is the partitioning phase, and the last term, R, is the remainder. Additionally, T is the time it takes to do a Cholesky factorization. The remainder consists of the time taken by Nektar++ to assemble the discretized system and for some other small and quick operations like computing the spectral radius. This time is just left as a remainder since it is completely dominated by the solve and partitioning phases. Another note is the time to solve for a single eigenpair is dominated by the time it takes to do a single parallel Cholesky factorization, therefore solving for $\frac{N}{P}$ eigenpairs takes about $\frac{N}{P}T(k, N, p)$ time.

To test the performance and accuracy of the solver, we ran experiments primarily on two computing clusters. The first is the University of Utah Center for High Performance Computing's Kingspeak cluster which accommodated jobs up to 12 nodes where each node is dual socket with Intel Xeon processors and 64GB of memory. For larger jobs, we used the Texas Advanced Computing Center's new Frontera computing system during its early access phase. Frontera is a powerful new cluster with 8,008 nodes with Intel Platinum 8280 processors (also dual socket) and 192 GB of memory per node. Utilizing this cluster allowed us to push the size of the jobs to a more extreme scale utilizing hundreds to thousands of cores.

4.1 Test Meshes

To test how the solver works on complex geometry, we used two test meshes, one in 2D and one in 3D. For the 2D mesh, we used a mesh of the Hanford site which can be seen in Figure 2 which has about 15k quadrilateral elements. For the 3D mesh, we used a mesh of an aorta that can be seen in Figure 7 which has about 24k tetrahedral elements. The number of degrees of freedom for both of these meshes can be increased by increasing the order of the basis elements in order to simulate larger and higher order examples.

4.2 Parallel Cholesky Factorization

Before we get into the solver performance, there is still the question of how to determine the configuration parameters of the two layers of parallelism, p and P. Specifically, how does the value of p affect the time it takes to do a parallel Cholesky factorization? For this, we selected a few simple sizes of cubes and factored the matrix $K - aM$ and measured the time taken for each value p. This experiment was

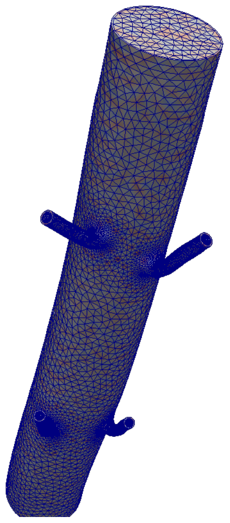

Fig. 7. Mesh of an aorta with about 24k tetrahedral elements.

run on the (Cluster 1) computing cluster and the results (plotted as parallel speedup) can be seen in figure 8. Additionally, we use the MUMPS [Amestoy et al. 2001] parallel Cholesky factorization library.

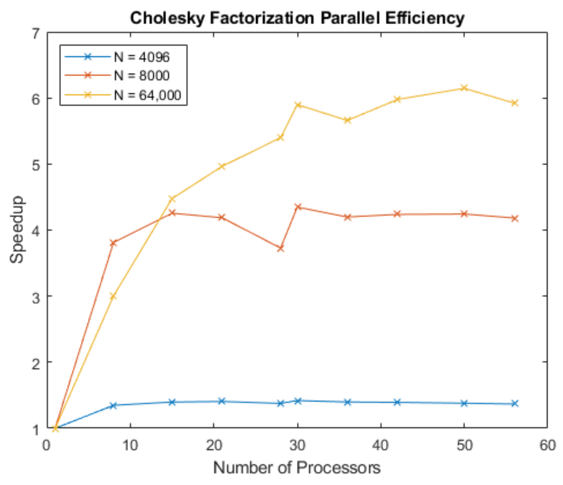

Fig. 8. Parallel speedup of Cholesky factorization for various N and p. Run on the (Cluster 1) cluster with 28 processors per node.

From this, it can be seen that the speedup from increasing p reaches diminishing returns fairly rapidly and is dependent on the size of the linear system. For very small problems, it makes sense to have many evaluators per node since the speedup plateaus at around 8 processors. However, our solver is geared towards very large scale problems and for these it makes more sense to use the full set of processors on a node. The default value then for p in our solver is to use all of the processors in a socket. This minimizes the communication between nodes and maximizes the number of tasks used per Cholesky factorization for large problems.

4.3 Strong Scalability

In order to verify that the solver phase actually scales as $\frac{N}{P}T(k, N, p)$ we performed a strong scaling experiment on a variety of 2D and 3D meshes with both simple and complex geometry with different numbers of degrees of freedom. Figure 9 shows the recorded elapsed time with respect to P and then Figure 10 shows the parallel speedup vs the ideal speedup.

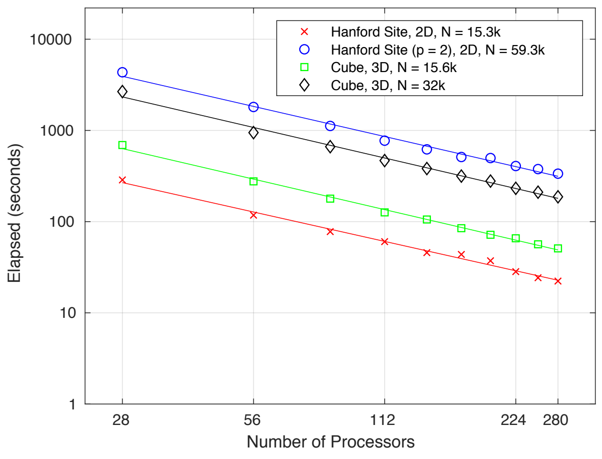

Fig. 9. Strong scaling for a collection of simple and complex 2D and 3D meshes on the (Cluster 1) computing cluster.

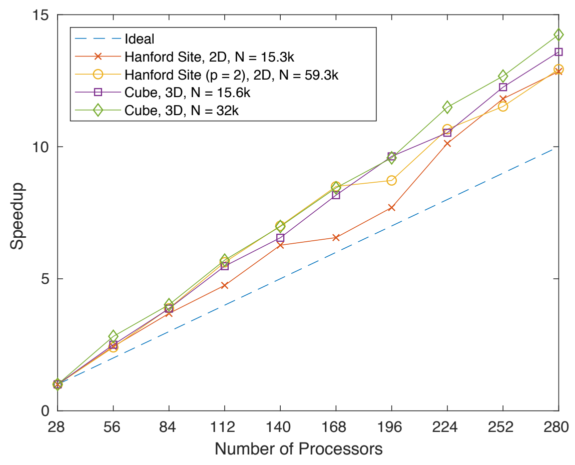

Fig. 10. Parallel speedup for a collection of simple and complex 2D and 3D meshes on the (Cluster 1) computing cluster.

For these small values of P, we observe super-linear speedup for all test cases. This is because the Krylov-Schur algorithm requires quite a lot more memory to compute N eigenpairs than $N/2$ and this memory requirement must exceed what can be cached so there is excessive memory movement for the $P = 1$ case that is mitigated when the problem is partitioned with larger P.

We recreated this experiment on the Frontera cluster on a larger complex 3D mesh (aorta) to see if this trend continued for larger values of P. The results of this can be seen in Figure 11. Eventually, as P keeps increasing, the parallel speedup does start to dip under the ideal linear speedup. However, while the parallel speedup drops below the ideal linear speedup for the Hanford site mesh at $P = 64$, the absolute run time with this value of P is a few seconds. At this point, there would be little benefit in increasing P.

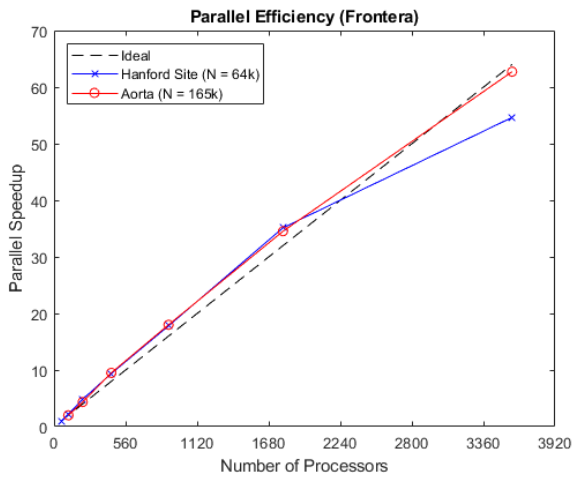

Fig. 11. Parallel speedup for Hanford site mesh and the aorta mesh on the Frontera computing cluster for P up to 64 nodes. A single node didn't have enough memory for the aorta so the baseline for the aorta mesh is $P = 2$. Parallel efficiency can be seen to drop off in the Hanford site once the noise in the greedy partitioning scheme dominates the magnitude of the optimal partitioning.

4.4 Weak Scaling

Since the key idea motivating this work is that if we need to compute the full spectrum for a very large mesh ($N \approx 500,000$), we need to be able to scale up the computing resources to produce the eigenbasis in a realistic amount of time. In order to see how many eigenpairs we could actually get with a feasible number of machines, we set up a range of problem sizes from 24k eigenpairs to half a million and increased P until we got a solve time under 10 minutes. This experiment was done on the Frontera cluster and the raw data can be seen in Table 1.

4.5 A brief note about eigenvalue solvers

In the introduction, we motivate the need for a solution to estimating the full spectrum of a sparse system and present Figure 1 to show how our method compares with the current implementation of SLEPc and ScaLAPACK. We did not include a comparison with other similar solvers such as the newest version of FEAST [Kestyn et al. 2016] or Eigenvalue Slicing Library (EVSL) [Li et al. 2018b] that both make use of spectrum slicing parallelism. All three of these libraries have spectrum slicing parallelism but stop short of providing a method for automatically partitioning the spectrum.

Table 1. Raw data for aorta weak scaling experiment.

Number of DoFs	**24k**	**165k**	**528k**
Number of Nodes	1	64	512
Number of Evaluators	2	128	512
Threads per Evaluator	28	28	56
Total Processes	56	3,584	28,672
Total Elapsed	**4m59s**	**4m23s**	**9m33s**
(Compute spectral radius)	3s	6s	20s
(Partition interval)	3s	25s	1m40s
(Solve)	4m44s	3m44s	7m14s
(Post-processing)	8s	8s	18s

While we present a comparison with SLEPc since it was the library that best fit our needs, our partitioning scheme is applicable to any spectrum slicing eigenvalue solver and is independent of the underlying solver implementation.

5 ACCURACY

To quantify the accuracy of this method, we use the two model problems mentioned in §2. The first is the diffusion equation with no forcing term. This represents the behavior of the homogenous solution to a more general diffusion equation. The second model problem represents the steady state solution as the homogeneous solution goes to 0. By separating the general diffusion equation into these two situations we can get a better idea of how the number of elements and basis order affects each regime in isolation.

The easiest way to test the accuracy is to use a simple domain like a square or a cube with known eigenfunctions that are easy to compute. For a square and cube, these are just the typical fourier modes. For these experiments, we also use homogeneous Dirichlet boundary conditions.

For the fractional homogeneous diffusion problem, we used the following initial conditions

$$u_0 = 2\sin(\pi x)\sin(\pi y) \tag{15}$$

which then have the exact solutions

$$u = 2e^{-\mu(2\pi^2)^{\alpha/2}t}\sin(\pi x)\sin(\pi y). \tag{16}$$

For the fractional Poisson problem, we used the same functions for the forcing,

$$f = 2\sin(\pi x)\sin(\pi y) \tag{17}$$

which have exact solutions

$$u = 2(2\pi^2)^{-\alpha/2}\sin(\pi x)\sin(\pi y). \tag{18}$$

Using these exact solutions, we computed approximate solutions for a variety of inputs. The first input is the order of the basis functions and we varied this value from 1 to 8. The second input is the number of elements in the mesh. For each of these, the solution error was computed for 11 equally spaced values of α from 0 to

2. The solution error for the diffusion equation was evaluated at $t = 0.4$ and $\mu = 1$.

The results of this equation for the Poisson equation can be seen in Figure 12 and the results for homogeneous diffusion can be seen in Figure 13. The main thing to note is that solution accuracy improves exponentially with respect to the order of the basis function. Increasing the number of elements improves the solution accuracy but not quite as drastically. This is because increasing the order improves accuracy at the low end of the spectrum where the input function "lives" but an input that has components in the high end of the spectrum might suffer with increased order.

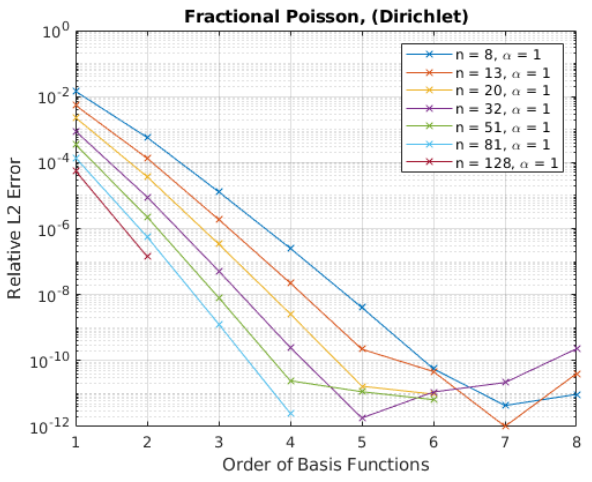

Fig. 12. Solution accuracy for fractional Poisson equation with respect to order of basis functions and number of elements.

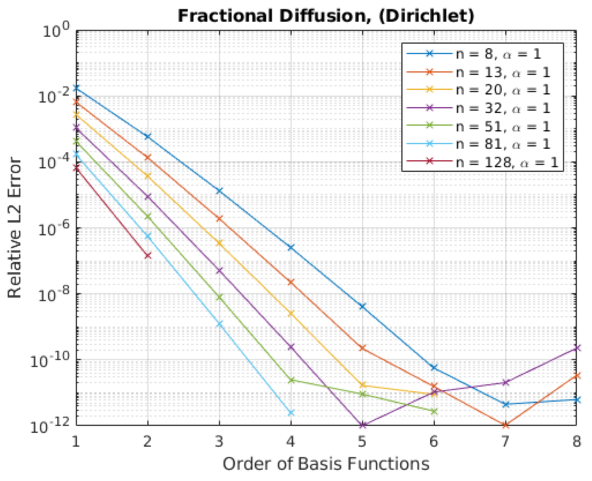

Fig. 13. Solution accuracy for fractional diffusion equation with respect to order of basis functions and number of elements.

While these plots show exponential convergence with respect to the order of the basis functions, the input functions are overly simple. Specifically, they are the first eigenfunctions of the Laplace operator and the approximation will reside almost entirely in the low end of the spectrum. Inputs that span more of the spectrum will likely require higher orders and number of elements in order to achieve the same level of accuracy.

6 CONCLUSION

Even though our eigenvalue solver can be used with any FEM library, for our experiments we have integrated our solver into the Nektar++ spectral/hp element framework [Cantwell et al. 2015]. We carefully matched the interface with the existing diffusion and Poisson solvers so that a researcher could use our fractional solvers without needing to change their workflow. With our scalable eigenvalue solver combined with this Nektar++ integration, we now have a scalable framework for solving fractional diffusion and Poisson equations. We hope that when this solver is pushed into the release branch of Nektar++ that it will be the first step in facilitating even more research into non-local operators on complex geometry.

Now that the framework is up and running for homogeneous boundary conditions, the next step is to include support for non-homogeneous BCs. According to [Lischke et al. 2018], there are a few approaches to handling these kinds of boundary conditions and further work will be in evaluating and implementing these methods.

Additionally, the largest performance bottleneck at the moment is the time it takes to solve each shifted $(K - aM)$ system using parallel Cholesky factorization. Since these systems are very sparse, we would like to utilize a preconditioned iterative method to approximate the solutions to these systems in a much smaller amount of time. One approach would be to use a multigrid based solver which would hopefully have the kind of convergence needed to beat a direct method like Cholesky factorization. We are looking into one such library that was developed for use with Nektar++ and further details of which can be found in [Rasouli et al. 2018].

Finally, applying the fractional operator to solve a given problem is still an $O(N^2)$ dense matrix-vector multiplication. We are currently experimenting with hierarchical matrix compression techniques to bring the time and space complexity of this operation to $O(N \log N)$. Our preliminary experiments in this direction have been positive and we hope to present this work in the near future.

ACKNOWLEDGMENTS

The first and third authors acknowledge the support of NSF (under DMS-1521748 and CCF-1704715) and ARO (MURI 00001271 subcontract with Brown University) respectively. The second authors acknowledges support from the Army Research Laboratory (ARL) under Cooperative Agreement Number W911NF-12-2-0023. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of ARL or the US Government. The US Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation herein. Additional thanks to Xiaoning Zheng and George Karniadakis of Brown University for discussions relating to the content of this paper.

REFERENCES

P. R. Amestoy, I. S. Duff, J. Koster, and J.-Y. L'Excellent. 2001. A Fully Asynchronous Multifrontal Solver Using Distributed Dynamic Scheduling. *SIAM J. Matrix Anal. Appl.* 23, 1 (2001), 15–41.

K. Assaleh and W. M. Ahmad. 2007. Modeling of speech signals using fractional calculus. In *2007 9th International Symposium on Signal Processing and Its Applications*. 1–4. https://doi.org/10.1109/ISSPA.2007.4555563

L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. 1997. *ScaLAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Daniele Boffi. 2010. Finite element approximation of eigenvalue problems. *Acta Numerica* 19 (2010), 1–120. https://doi.org/10.1017/S0962492910000012

C.D. Cantwell, D. Moxey, A. Comerford, A. Bolis, G. Rocco, G. Mengaldo, D. De Grazia, S. Yakovlev, J.-E. Lombard, D. Ekelschot, B. Jordi, H. Xu, Y. Mohamied, C. Eskilsson, B. Nelson, P. Vos, C. Biotto, R.M. Kirby, and S.J. Sherwin. 2015. Nektar++: An open-source spectral/hp element framework. *Computer Physics Communications* 192 (2015), 205 – 219. https://doi.org/10.1016/j.cpc.2015.02.008

Yuyao Chen, Alfredo Fiorentino, and Luca Dal Negro. 2019. A fractional diffusion random laser. *Scientific Reports* 9, 1 (2019), 1–14. https://doi.org/10.1038/s41598-019-44774-3

Nicole Cusimano, Alfonso Bueno-Orovio, Ian Turner, and Kevin Burrage. 2015. On the Order of the Fractional Laplacian in Determining the Spatio-Temporal Evolution of a Space-Fractional Model of Cardiac Electrophysiology. *PLOS ONE* 10, 12 (12 2015), 1–16. https://doi.org/10.1371/journal.pone.0143938

Nicole Cusimano, Kevin Burrage, and Pamela Burrage. 2013. Fractional models for the migration of biological cells in complex spatial domains. *ANZIAM Journal* 54, 0 (2013), 250–270. https://doi.org/10.21914/anziamj.v54i0.6283

Anton Evgrafov and José C. Bellido. 2018. From nonlocal Eringen's model to fractional elasticity. arXiv:1806.03906 [math.AP]

Alejandro Guerrero and Miguel Angel Moreles. 2015. On the numerical solution of the eigenvalue problem in fractional quantum mechanics. *Communications in Nonlinear Science and Numerical Simulation* 20, 2 (2015), 604 – 613. https://doi.org/10.1016/j.cnsns.2014.06.013

Bruce Ian Henry, T. A. M. Langlands, and Peter Straka. 2010. An Introduction to Fractional Diffusion.

Vicente Hernandez, Jose E. Roman, and Vicente Vidal. 2005. SLEPc: A Scalable and Flexible Toolkit for the Solution of Eigenvalue Problems. *ACM Trans. Math. Softw.* 31, 3 (Sept. 2005), 351–362. https://doi.org/10.1145/1089014.1089019

Sverre Holm and Sven Peter Näsholm. 2011. A causal and fractional all-frequency wave equation for lossy media. *The Journal of the Acoustical Society of America* 130, 4 (2011), 2195–2202. https://doi.org/10.1121/1.3631626 arXiv:https://doi.org/10.1121/1.3631626

J. Kestyn, V. Kalantzis, E. Polizzi, and Y. Saad. 2016. PFEAST: A High Performance Sparse Eigenvalue Solver Using Distributed-Memory Linear Solvers. In *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*. 178–189.

Sunil Kumar, Devendra Kumar, and Jagdev Singh. 2014. Numerical computation of fractional Black–Scholes equation arising in financial market. *Egyptian Journal of Basic and Applied Sciences* 1, 3 (2014), 177 – 183. https://doi.org/10.1016/j.ejbas.2014.10.003

Nick Laskin. 2002. Fractional Schrödinger equation. *Physical review. E, Statistical, nonlinear, and soft matter physics* 66 5 Pt 2 (2002), 056108.

Sergei Levendorskii. 2004. PRICING OF THE AMERICAN PUT UNDER LÉVY PROCESSES.

Ruipeng Li, Yuanzhe Xi, Lucas Erlandson, and Yousef Saad. 2018a. The Eigenvalues Slicing Library (EVSL): Algorithms, Implementation, and Software. arXiv:1802.05215 [math.NA]

Ruipeng Li, Yuanzhe Xi, Lucas Erlandson, and Yousef Saad. 2018b. The Eigenvalues Slicing Library (EVSL): Algorithms, Implementation, and Software. arXiv:1802.05215 [math.NA]

Anna Lischke, Guofei Pang, Mamikon Gulian, Fangying Song, Christian Glusa, Xiaoning Zheng, Zhiping Mao, Wei Cai, Mark M. Meerschaert, Mark Ainsworth, and George Em Karniadakis. 2018. What Is the Fractional Laplacian? arXiv:1801.09767 [math.NA]

Richard L. Magin. 2010. Fractional calculus models of complex dynamics in biological tissues. *Computers & Mathematics with Applications* 59, 5 (2010), 1586 – 1593. https://doi.org/10.1016/j.camwa.2009.08.039 Fractional Differentiation and Its Applications.

Martin J. Mohlenkamp. 1999. A fast transform for spherical harmonics. *Journal of Fourier Analysis and Applications* 5, 2 (01 Mar 1999), 159–184. https://doi.org/10.1007/BF01261607

Edoardo Di Napoli, Eric Polizzi, and Yousef Saad. 2013. Efficient estimation of eigenvalue counts in an interval. arXiv:1308.4275 [cs.NA]

Fernando A. Oliveira, Rogelma M. S. Ferreira, Luciano C. Lapas, and Mendeli H. Vainstein. 2019. Anomalous Diffusion: A Basic Mechanism for the Evolution of Inhomogeneous Systems. *Frontiers in Physics* 7 (2019), 18. https://doi.org/10.3389/fphy.2019.00018

M. Rasouli, V. Zala, R. M. Kirby, and H. Sundar. 2018. Improving Performance and Scalability of Algebraic Multigrid through a Specialized MATVEC. In *2018 IEEE High Performance extreme Computing Conference (HPEC)*. 1–7. https://doi.org/10.1109/HPEC.2018.8547580

Vasily E. Tarasov and Elias C. Aifantis. 2018. On Fractional and Fractal Formulations of Gradient Linear and Nonlinear Elasticity. arXiv:1808.04452 [physics.class-ph]