# PFEAST: A High Performance Sparse Eigenvalue Solver Using Distributed-Memory Linear Solvers

James Kestyn, Vassilis Kalantzis, Eric Polizzi, and Yousef Saad

August 2016

EPrint ID: 2016.2

Department of Computer Science and Engineering
University of Minnesota

# PFEAST: A High Performance Sparse Eigenvalue Solver Using Distributed-Memory Linear Solvers

James Kestyn*, Vasileios Kalantzis†, Eric Polizzi*, Yousef Saad†
*Electrical and Computer Engineering Department, University of Massachusetts, Amherst, MA, U.S.A.
†Computer Science and Engineering Department, University of Minnesota, Minneapolis, MN, U.S.A.

***Abstract*—The FEAST algorithm and eigensolver for interior eigenvalue problems naturally possesses three distinct levels of parallelism. The solver is then suited to exploit modern computer architectures containing many interconnected processors. This paper highlights a recent development within the software package that allows the dominant computational task, solving a set of complex linear systems, to be performed with a distributed memory solver. The software, written with a reverse-communication-interface, can now be interfaced with any generic MPI linear-system solver using a customized data distribution for the eigenvector solutions. This work utilizes two common "black-box" distributed memory linear-systems solvers (Cluster-MKL-Pardiso and MUMPS), as well as our own application-specific domain-decomposition MPI solver, for a collection of 3-dimensional finite-element systems. We discuss and analyze how parallel resources can be placed at all three levels simultaneously in order to achieve good scalability and optimal use of the computing platform.**

## I. Introduction

Eigenvalue problems are widely used across a diverse range of high performance computing applications. A generalized problem is defined by two $n \times n$ matrices $A$ and $B$, with the set ($A$, $B$) known as the matrix pencil. The eigenvalues $\Lambda$ and eigenvectors $X$ are non-trivial solutions to

$$AX = BX\Lambda. \tag{1}$$

The problem is called 'standard' if $B$ reduces to the identity matrix or 'generalized' otherwise. In spite of the enormous progress that has been made over the last few decades in algorithms and software packages that compute the solution to large sparse eigenvalue problems, the current state-of-the-art methods are facing new challenges for achieving ever higher levels of efficiency, accuracy and performance on modern parallel architectures. In particular, traditional methods suffer from the orthogonalization of a very large basis when many eigenpairs are computed. In this case, a divide-and-conquer approach that can compute wanted eigenpairs by parts, becomes mandatory since 'windows' or 'slices' of the spectrum can be computed independently of one another and orthogonalization between eigenvectors in different slices is no longer necessary. In this framework, all the resulting subintervals are called interior eigenvalue problems, in the sense that they involve large blocks of eigenpairs located anywhere inside the spectrum.

The FEAST algorithm [27] and associated software package [28], [9] is an accelerated subspace iterative technique for computing interior eigenpairs that makes use of a rational filter obtained from an approximation of the spectral projector. FEAST can be applied for solving both standard and generalized forms of Hermitian or non-Hermitian problems, and belongs to the family of contour integration eigensolvers [32], [33], [3], [14], [15], [4]. Once a given search interval is selected, FEAST's main computational task consists of a numerical quadrature computation that involves solving independent linear systems along a complex contour. The algorithm can exploit natural parallelism at three different levels: (i) search intervals can be treated separately (no overlap), (ii) linear systems can be solved independently across the quadrature nodes of the complex contour, and (iii) each complex linear system with multiple right-hand-sides can be solved in parallel. Within a parallel environment, the algorithm complexity becomes then directly dependent on solving a single linear system.

The FEAST numerical library offers 'black-box' reverse communication interfaces (RCI) which are both matrix format and linear system solver independent, and can then be fully customized by the end users to allow maximum flexibility for their applications. In addition, FEAST offers the following set of desirable features: (i) high-robustness and a well defined convergence rate, (ii) all multiplicities naturally captured, (iii) no explicit orthogonalization procedure on eigenvectors, and (iv) a reusable subspace when solving a series of related eigenproblems. Consequently, the software package has been very well received by application developers, especially in the electronic structure community. A common technique to calculate the electronic structure and ground-state properties of molecules is Density Functional Theory (DFT) [20], where many eigenvalue problems must be solved within a self consistent loop. The FEAST algorithm is an ideal candidate to parallelize this step, speeding up the time-to-solution for these calculations and allowing for the investigation of very large molecules containing thousands of atoms.

So far, the FEAST software has been limited to the use of shared-memory system solvers (at the third level of parallelism). In this paper we extend the software to operate on distributed memory platforms and interface the fully parallel version of FEAST (PFEAST) with three different MPI linear-system solvers: (i) Cluster-MKL-Pardiso [17], (ii) MUMPS [1], and (iii) our own custom domain-decomposition solver for electronic structure calculations [24]. Three separate levels of communication must be managed within the software kernel and the multilevel parallel capabilities of the eigensolver result in a trade-off between memory and performance. Parallel resources can be placed at all three levels simultaneously in order to achieve good scalability and optimal use of the computing platform. This paper aims to highlight the flexibility of the eigensolver and describe how the new PFEAST kernel can be interfaced with different MPI linear-system solvers using specific data distributions for input matrices and right-hand-side vectors. We then benchmark the eigensolvers performance with the three different solvers, showcase its scalability at each level of parallelism, and discuss how to optimally distribute parallel resources.

## II. The FEAST Algorithm

The FEAST algorithm utilizes spectral projection and subspace iteration to obtain selected interior eigenpairs. A Rayleigh-Ritz procedure is used to project matrices $A$ and $B$ onto a reduced search subspace to form matrices

$$A_q = Q^H AQ \quad \text{and} \quad B_q = Q^H BQ. \tag{2}$$

Approximate eigenvalues $\tilde{\Lambda}$ and eigenvectors $\tilde{X}$ of the original system (i.e. Ritz-values and Ritz-vectors) can then be recovered from the solutions of the much smaller eigenvalue problem

$$A_q W_q = B_q W_q \Lambda_q \tag{3}$$

as

$$\tilde{X} = QW_q \quad \text{and} \quad \tilde{\Lambda} = \Lambda_q. \tag{4}$$

Initializing $\tilde{X}_m$ as a set of $m$ random vectors and obtaining $Q_m$ after $QR$ factorization of $\tilde{X}_m$, results in a standard subspace iteration (i.e. power method) that converges linearly toward the dominant eigenpairs [31]. Many other sophisticated Krylov-based methods have also been developed to improve the convergence rate for the calculation of selected smallest, largest or interior eigenpairs [22], [19], [6], [36]. The subspace iteration technique, in turn, can be efficiently used for solving the interior eigenvalue problem when it is combined with filtering which aims to improve the convergence by increasing the gap between wanted eigenvalues and unwanted ones. It is well known that Ritz-pairs ($\tilde{X}_m$, $\tilde{\Lambda}$) converge toward the true eigenpairs ($X_m$, $\Lambda$) at a rate determined by the filter [31], [26], [38]. In theory, the ideal filter for the Hermitian problem would act as a projection operator $X_m X_m^H B$ onto the subspace spanned by the eigenvector basis, which can be expressed via the Cauchy integral formula:

$$X_m X_m^H B = \oint_\Gamma dz(zB - A)^{-1}B, \tag{5}$$

where the eigenvalues associated with the $B$-orthonormal eigenvectors $X_m$ are located within a given search interval delimited by any closed curve $\Gamma$.

In practice, the spectral projector must be approximated using a quadrature rule using $n_e$ integration nodes and weights $\{(z_j, \omega_j)\}_{j=1,\dots,n_e}$ i.e.

$$Q_{m_0} = \sum_{j=1}^{n_e} \omega_j (z_j B - A)^{-1} B\tilde{X}_{m_0}, \tag{6}$$

where we also consider a search subspace of size $m_0 \geq m$. The computation of $Q_{m_0}$ amounts to solving a set of $n_e$ complex shifted linear-systems

$$(z_j B - A)Q_{m_0}^{(j)} = B\tilde{X}_{m_0} \quad \text{with} \quad Q_{m_0} = \sum_{j=1}^{n_e} \omega_j Q_{m_0}^{(j)}. \tag{7}$$

**Algorithm 1** The FEAST Algorithm

```
input: $A$, $B$, $\tilde{X}_{m_0}$, $\{(z_j, \omega_j)\}_{1,\dots,n_e}$, $\epsilon$
while ( $||A\tilde{X}_m - B\tilde{X}_m\Lambda_m|| > \epsilon$ ) do
    $Q_{m_0} = 0$
    for ( $j = 0; j < n_e; j = j + 1$ ) do
        $Q_{m_0}^{(j)} \leftarrow (z_j B - A)^{-1} B\tilde{X}_{m_0}$
        $Q_{m_0} \leftarrow Q_{m_0} + \omega_j Q_{m_0}^{(j)}$
    end for
    $A_q = Q^H AQ \quad B_q = Q^H BQ$
    Solve $A_q W_q = B_q W_q \Lambda_q$
    $\tilde{X}_{m_0} = Q_{m_0} W_q \quad \tilde{\Lambda} = \Lambda_q$
end while
output: $\tilde{X}_m$, $\tilde{\Lambda}_m$
```

This matrix $Q_{m_0}$ is then used as the Rayleigh-Ritz projection operator to form reduced matrices $A_q$ and $B_q$ of (2). If the exact spectral projector was known, solving the reduced eigenproblem in (3) will produce the exact eigenvalues $\tilde{\Lambda} = \Lambda_q = \Lambda$ and eigenvectors $\tilde{X} = QW_q = X$. However, since it is only approximated, the Ritz-values $\Lambda_q$ and updated Ritz-vectors $\tilde{X}$ are only an approximation to the true eigenpairs. Subspace iteration will then, in effect, tilt the subspace spanned by columns of $\tilde{X}$ toward the desire eigenspace. At convergence we will obtain $\tilde{X} = Q = X$ and $\tilde{\Lambda} = \Lambda$. The general outline can be seen in Algorithm 1 for computing $m$ eigenpairs in a given search interval. The input $\tilde{X}$ can be chosen as a set of $m_0$ random vectors or a previously calculated solution to a closely related problem.

The purpose of this section is not to provide a thorough understanding of the FEAST algorithm, but to give a general idea of the algorithmic steps involved. A detailed numerical analysis can be found in [38]. Additional information regarding the application of FEAST to non-symmetric and non-Hermitian systems is available in [18], [37], [21], [40]. In particular, we would like to emphasize that the main computational procedure within the algorithm is solving the set of complex linear-systems in (7).

## III. Three Levels of MPI Parallelism

Inherent to the FEAST algorithm are three separate levels of parallelism mentioned in Section I that we will from now denote as **L1**, **L2** and **L3**. The current release of the software package - FEASTv3.0 - offers MPI [12] only for the first two levels **L1** and **L2**. The upcoming release - FEASTv4.0 - will place MPI also at the third level **L3**, allowing the linear systems solutions to be solved with a distributed memory solver.

The first level **L1** can be used to distribute the eigenvalue spectrum if a large number of eigenpairs are needed.

In practice, can be used to limit the search subspace size $m_0$ and hence the number of right-and-sides for the linear systems. If $m_0$ is composed of at most a thousand vectors, the $m_0 \times m_0$ reduced eigenvalue problem in (3) can be solved using standard dense methods on a single node using LAPACK [2]. For larger values of $m_0$ a parallel treatment of the reduced system (e.g. with ScaLAPACK [5]) could also be possible, but is not considered in this work. To efficiently utilizing the **L1** level of parallelism one must ensure each interval contains roughly the same number of eigenvalues (uniform slicing), possibly by using the fast stochastic estimator [8] already incorporated within FEASTv3.0. The Zolotarev quadrature can then help to provide a uniform convergence rate between different contours [13]. Alternatively, an economical "augmented subspace approach" has recently been proposed to achieve the same goal [11].

For a given search interval, each linear system in (7) can be calculated independently with the second level of parallelism **L2**. This should result in close to linear scaling. However, this level is constrained by the number of nodes used in the quadrature rule. The addition of quadrature nodes will improve the approximate integration and increase the convergence rate, but with a direct solver FEAST usually converges in few iterations ($\sim 4$) using only 8 to 16 nodes. This level of parallelism is limited since the inclusion of more than 16 nodes may not significantly improve the performance (i.e. decrease the number of FEAST iterations).

At level **L3**, each linear system can be solved in parallel. Parallelism for the linear system solutions has previously been introduced via threading within LAPACK routines for dense systems, PARDISO [34], [16] for sparse systems, and SPIKE-SMP [25], [35] for banded ones. It is this level that we have addressed in this work: upgrading the FEAST computational kernel to allow for a distributed memory linear-system solver. The additional level of threaded parallelism will now take place within level **L3** and it would be inherent to the specific MPI system solver implementation.

The second and third levels of parallelism overlap and must be managed. This results in two separate MPI-Communicators shown in Figure 1. We refer to the collection of MPI processes within an MPI communicator as a "Communication-World". At the second level, the **L2** communicator (defining each **L2**-Communication-World) is a carry-over from the previous FEAST distribution and specifies the mapping between MPI processes and quadrature nodes (i.e. linear systems). Each MPI process in **L2** maps directly to a set of quadrature nodes. Ideally, the number of members within each **L2** communicator will be equal to the number of quadrature nodes and each linear-system is solved in parallel. In this case, using a direct solver, each matrix factorization $(z_i B - A)$ must be computed only once since subsequent PFEAST iterations solve a linear system at the same complex pivot, but with an updated set of right-hand-sides. The set of processes at the third level of parallelism (i.e. **L3**-Communication-World) is to be used by the distributed memory solver. In Figure 1 there are six total MPI processes **P-0** through **P-5**. **L2** communicator (MPI) ranks **F-0**, **F-1** and **F-2** map to contour integration nodes $z_0$, $z_1$ and $z_2$, where a linear system $(z_i B - A)Q = BY$ must be solved. Each integration node then owns exactly two **L3** MPI processes **S-0** and **S-1** to be used by the distributed memory solver.

Dividing parallel resources among the second and third levels of parallelism results in a trade-off between memory and performance. The second level represents ideal linear scaling, since each linear-system can be solved independently. However, it also requires more memory. Each cluster of **L3** MPI processes (within an **L3**-Communication-World) will require a copy of the matrix. Placing more MPI processes at the third level of parallelism, in turn, will reduce the amount of memory required to store the matrix and eigenvector solutions (since they are distributed across the **L3** processes), which could become essential to many large-scale applications.

If the amount of memory required to store the matrix and eigenvector solutions is too large, it can be reduced using the first and third levels of parallelisms **L1** and **L3**. The **L1** level of parallelism subdivides the FEAST search interval resulting in fewer calculated eigenpairs per subinterval. As shown in Figure 2, this reduces the number columns in the eigenvector matrix per MPI process. Additional MPI processes at the **L3** level will allow the matrix and eigenvector solutions to be distributed by row. If both **L1** and **L3** are used each MPI process will contain a subset of columns and rows as seen in the bottom right of Figure 2.

All MPI solvers must adhere to a specific pre-defined data distribution, which can differ for each implementation. The solver defines a data distribution for both the matrix and rhs/solution vectors. PFEAST will require a predefined data distribution for its kernel as well. The distribution format for PFEAST is defined for the eigenvectors $X$ (as well as $Q$ and Ritz-vectors $\tilde{X}$), and is independent of the distribution format of matrices $A$ and $B$. The kernel then operates directly on the distributed vectors.

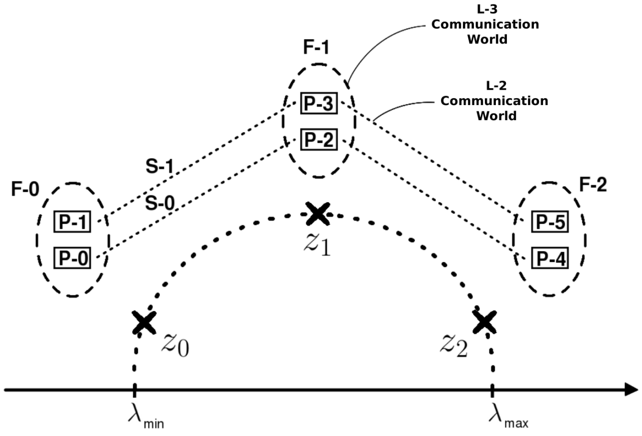

Fig. 1. Overlapping MPI communicators for PFEAST. We refer to the collection of MPI processes within an MPI communicator as a "Communication-World". If the first level of parallelism was also used, this picture would then represent a single **L1**-Communication-World.

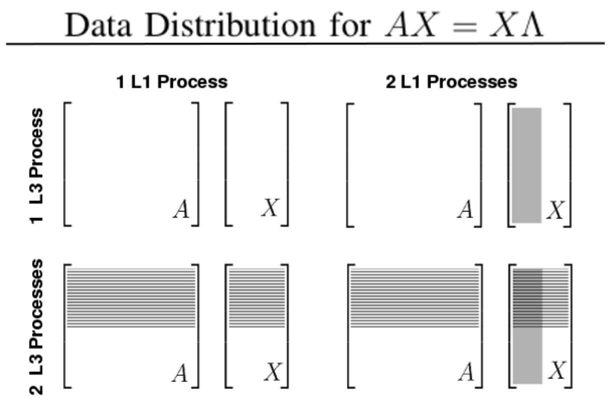

Fig. 2. Data distribution of input matrix $A$ and eigenvector solutions $X$ for different configurations of parallel resources. Additional MPI processes at the **L1** and **L3** levels reduce the memory required to store the matrix and solution. The **L1** level subdivides the PFEAST search interval and results in fewer number of eigenvectors calculated per node. Likewise, the **L3** level can be used to distribute $X$ by row and $A$ (as well as $B$ for generalized problems) by the distributed memory solver format. MPI processes at the **L2** level would result in additional copies of both $A$ and $X$.

The eigenvectors data distribution has been defined to be 1-dimensional and by row. Each MPI process within an **L3** communicator stores and operates on a specific subset of rows and all corresponding columns. The choice of a 1-dimensional distribution can be justified by the fact that PFEAST calculates only a subset of $m_0$ eigenvectors; the number of rows $n$ in the eigenvector matrix will, in general, be much larger than the number of columns (i.e. $n >> m_0$). Additionally, the eigenvector columns can be distributed independently of the PFEAST kernel with the first level of parallelism **L1**. The data distribution will be the same for the right-hand-side vector supplied by PFEAST and linear-system solutions returned to PFEAST as they are directly related to the eigenvectors and have the same matrix dimensions. Since the MPI solver must operate on rhs/solution vectors in its own format, a reordering step with communication between the MPI processes within each **L3** communicator could be necessary.

The distribution of the eigenvectors is required to be the same for each of the **L3** communicators; i.e. the same number of MPI processes is applied to each linear-system and equivalent processes for different **L2** communicators must correspond to the same eigenvector partition. Both the linear-system and eigenvectors solutions are, in general, dense and will be stored in a matrix using the following 1-dimensional distribution:

$$Q_k = [\ q_1^T,\ \ q_2^T,\ \ \ldots,\ \ q_s^T\ ]^T. \tag{8}$$

Here the $k^{th}$ **L2** (MPI) rank has its solution vector distributed across $s$ **L3** MPI processes.

The data distribution will match for the members of each **L2** communicator and the distributed Ritz-vectors $Q$ can be found through an all-reduce operation; i.e. scaling each linear-system solution by the corresponding integration weight and summing the results. The PFEAST kernel then operates only on a local subset of rows $q_i$ within $Q_k$. This row range is specified in two entries within the PFEAST parameter array and does not need to be equal for each **L3** MPI process. Users can implement their own matrix multiplication routine or linear system solver as long as the result is placed back into the correct position within the 1-dimensionally distributed $Q_k$ matrix.

It happens that, if the second level of parallelism **L2** is used, a copy of $Q$ will be stored for each **L2** (MPI) rank. Subsequently, at convergence multiple copies of the eigenvectors $X = \tilde{X} = Q$ will also be stored across the **L2** communicator (i.e. for each **L3**-Communication-World). However, since PFEAST must compute two inner-products of the form $A_q = Q^H AQ$ and $B_q = Q^H BQ$ further parallelization can be achieved. Moreover, since all columns of $Q$ are known across the **L2** (MPI) ranks, only a small $m_0 \times m_0$ communication is necessary.

After the contour integration has been evaluated to find $Q$, all MPI processes from **L2** and **L3** become available. Both matrix multiplications $Y = \{AQ, BQ\}$ (outside kernel) and $\{A_q, B_q\} = Q^H Y$ (inside kernel) can take advantage of additional **L2** MPI processes. To make use of all MPI processes $Q$ is subdivided and given a 2-dimensional decomposition. The matrix $Q$, which has already been distributed by row across the $s$ MPI processes within an **L3** communicator, is now further distributed by column across the $f$ **L2** processes. This results in the 2-dimensional decomposition

$$Q = \begin{bmatrix} q_{11} & q_{12} & \cdots & q_{1f} \\ q_{21} & q_{22} & \cdots & q_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ q_{s1} & q_{s2} & \cdots & q_{sf} \end{bmatrix}. \tag{9}$$

Each block $q_{ij}$ of matrix $Q$ matches to exactly one MPI process. However, all columns of the matrix $q_i = [q_{i1}, ..., q_{if}]$ are known for each MPI rank in the **L2** communicator. That is, each MPI process within an **L2** communicator has in memory all the columns of the 1-dimensionally distributed $q_i$, but will only be performing the multiplication for one piece $q_{ij}$ of the 2-dimensional distribution. The multiplication result for each $q_{ij}$ must be placed back into the correct location within the matrix $Y$. After the multiplication is performed on all $s \times f$ MPI processes, the matrix $Y = \{AQ, BQ\}$ will have the same decomposition as (9) except that each block is known only on a single MPI process.

The matrix multiplication is needed for three separate stages of the algorithm and the operations within the kernel, although transparent to the user, differ slightly. First, it is used to compute the Rayleigh-Ritz projection for $\{A_q, B_q\}$. In this case, the kernel leaves $Y$ in its 2-dimensional distribution and the matrix multiplication has the form

$$A_q = Q^H AX = Q^H Y =$$

$$[\ q_1^H\ \ q_2^H\ \ \ldots\ \ q_s^H\ ] \times \begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1f} \\ y_{21} & y_{22} & \cdots & y_{2f} \\ \vdots & \vdots & \ddots & \vdots \\ y_{s1} & y_{s2} & \cdots & y_{sf} \end{bmatrix}, \tag{10}$$

where each local chunk $y_{ij}$ is known on a single MPI process only. The vector $q_i$, however, is known for all $f$ MPI processes with the **L2** communicator. When performing the dense multiplication $\{A_q, B_q\} = Q^H Y$ inside the kernel each $y_{i1}, ..., y_{if}$ must be multiplied with $q_i^H$. By storing the result of each $q_i^H y_{ij}$ to the correct location within $\{A_q, B_q\}$, only a small $m_0 \times m_0$ communication is necessary. Additionally, both $AQ$ and $BQ$ are needed in the computation of the eigenvector residuals. The result $Y = \{AQ, BQ\}$ is again left in its 2-dimensional distribution. This result is independent for each eigenvector and therefore for each MPI rank within an **L2** communicator. And, since the $L_1$ norm is used, the final residual for each vector can be found by summing the scalar result within each of the **L3**-Communication-Worlds. After the residuals are determined, the final (scalar) values are communicated among all MPI processes. Also required is the right-hand-side for the linear-systems, which must be updated as $Y = BQ$ at the beginning of each PFEAST iteration. Here $y_i$, the 1-dimensionally distributed piece of $Y$, is required for each MPI process within the **L2** communicator and a reduction operations must be performed across the **L2** communicator, summing the results for each **L3**-Communication-World.

## IV. FEAST WITH A DISTRIBUTED-MEMORY SOLVER

Solving the linear-systems in (7) will be the dominant computational procedure performed in the eigenvalue calculation. Different "black-box" direct distributed memory solvers have been integrated with the PFEAST kernel. An interface has been created for two sparse direct solvers: the cluster version [17] of PARDISO [34] within the Intel Math Kernel Library [16] and MUMPS [1]. A banded interface has also been created for the SPIKE-MPI [30], although we do not present any results for banded matrices. Additionally, we highlight the versatility of the PFEAST kernel by describing how to integrate a Schur complement type solver.

Each interface requires a matrix-matrix multiplication routine specific to the distribution format accepted by the solver. To interface these solvers with the PFEAST kernel, linear-system solution and matrix multiplication results must be placed back into the correct row corresponding to the PFEAST distribution for the eigenvectors. Cluster-MKL-PARDISO offers multiple options for distributing the matrix and rhs/solution vector. The matrix and vectors can be distributed 1-dimensionally by row across the **L3** MPI processes or stored locally on the head node. We have chosen to distribute both the matrix and the right-hand-side vector for the linear-system solver in order to match the eigenvector distribution. There is then no need for a reordering step since the PFEAST data distribution matches with the solver. MUMPS also offers multiple options for the matrix distribution and we have chosen to manually distribute the matrix by row to be consistent. However, MUMPS requires the rhs/solution to be fully constructed on the head node and an additional communication is then needed before and after the solve stage.

The PFEAST kernel can also be integrated within a domain decomposition framework. One approach is to divide the matrix $A$ using a graph partitioner. The reordering will create a $2 \times 2$ block matrix

$$A = \begin{bmatrix} C & E \\ E^T & D \end{bmatrix}, \tag{11}$$

where $C$ contains the independent blocks and $E$ and $D$ contain the connections between them. Linear-systems possessing this block form can be solved with a Schur complement approach. The procedure arises by row reducing the matrix equation $AX = Y$ to an upper block format, resulting in the equation

$$\begin{bmatrix} C & E \\ 0 & S \end{bmatrix} \begin{bmatrix} X_l \\ X_e \end{bmatrix} = \begin{bmatrix} Y_l \\ Y_e - E^T C^{-1} Y_l \end{bmatrix}, \tag{12}$$

where $X_l$ and $X_e$ refer to "local" and "external" pieces of the vector. The quantity $S := (D - E^T C^{-1} E)$ is known as the Schur complement. The solution to the linear-system can be obtained in a three step process.

(i) Solve: $CT = Y_l$
(ii) Solve: $SX_e = Y_e - E^T T$
(iii) Solve: $CX_l = Y_l - E^T X_e$

The first and third steps can be trivially parallelized since $C$ has a block diagonal structure. The difficulty arises in steps (ii). In particular the formation of the Schur complement is non-trivial and requires an inversion of the $C$ matrix, or equivalently the solution to $CT = E$. Also, the Schur matrix $S$ is not block diagonal as was the case for $C$. Efficiently solving the Schur complement equation of step (ii) in parallel for $X_e$ is much more demanding and requires communication between MPI processes.

With PFEAST we are interested in a series of linear-systems in (7). The matrix $(z_j B - A)$ can be reordered to posses a block form in (11). The goal of domain-decomposition is then to obtain the solution to this linear systems in parallel using the Schur complement. The matrix reordering will implicitly partition vectors $Q$ and $Y = B\tilde{X}$ (PFEAST inputs/outputs) into "local" and "external" pieces

$$Q \equiv \begin{bmatrix} Q_l \\ Q_e \end{bmatrix} \quad and \quad B\tilde{X} = Y \equiv \begin{bmatrix} Y_l \\ Y_e \end{bmatrix}. \tag{13}$$

It is then convenient to think of the "local" pieces $Q_l$ and $Y_l$ separately from the "external" $Q_e$ and $Y_e$.

The matrix $C$ has a block diagonal form

$$\begin{bmatrix} C_1 & & & \\ & C_2 & & \\ & & \ddots & \\ & & & C_p \end{bmatrix} \begin{bmatrix} q_l^1 \\ q_l^2 \\ \vdots \\ q_l^p \end{bmatrix} = \begin{bmatrix} y_l^1 \\ y_l^2 \\ \vdots \\ y_l^p \end{bmatrix}, \tag{14}$$

and the solution to the linear-system $CQ_l = Y_l$ can be obtained in parallel. Each linear system $C_i q_l^i = y_l^i$ can be solved independently on a different **L3** MPI process. The vectors $Q_l$ and $Y_l$ should also be distributed across the MPI processes in the same manner so that $C_i$, $q_l^i$ and $y_l^i$ all exist on the same node.

The Schur complement solve in step (ii) should also be computed in parallel by distributing the vectors $Q_e$ and $Y_e$ among the **L3** MPI processes. The PFEAST kernel is independent of the ordering. Each piece of $Q$ (resp. $Y$), local to the $i^{th}$ **L3** MPI process would contain a subset of rows from both $Q_l$ (resp. $Y_l$) and $Q_e$ (resp. $Y_e$):

$$q_i = \begin{bmatrix} q_l^i \\ q_e^i \end{bmatrix} \quad with \quad Q = \begin{bmatrix} q_1 \\ \vdots \\ q_s \end{bmatrix}. \tag{15}$$

The "local" components $y_l$ would be extracted from the PFEAST work array and operated on for steps (i) and (iii). The solution $q_l$ must then be placed back into the work array at the correct location (i.e to match (15)). Step (ii) would also have to extract a subset of the "external" points from the right-hand-side vector and place the solution back into the correct location.

## V. All-electron DFT Application

This parallel version of FEAST has been integrated with our own finite-element electronic-structure code [7], [24], [10] used to compute ground-state and excited-state properties of molecules through DFT and time dependent DFT (TDDFT). Ground-state DFT calculations require many eigenvalue problems to be solved in a self consistent loop. This is, along with the solution to the Poisson equation, the most computationally demanding step in the procedure. A highly parallel eigensolver can significantly speed up the time-to-solution for these problems and allow for the investigation of very large molecules comprised of many thousands of atoms. The results presented in the next section show the scalability of the eigenvalue computation for a single iteration of the self-consistent scheme used in DFT. The time-to-solution, ignoring the Poisson equation, would then depend mainly on the number of iterations required to reach convergence.

Our all-electron code does not use pseudopotentials and explicitly includes the effect of core electrons. There are many more points directly surrounding the atoms than in the region between in order to capture high-frequency variation of the core wave functions. A domain-decomposition approach is used to separate dense regions of points directly surrounding the atoms [39]. The idea is to create a separate distinct finite-element mesh for each atom [23]. The atomic regions are then connected through an additional "interstitial" mesh. This gives rise to the muffin-tin overlapping (conformal) domain-decomposition in Figure 3 where the atomic regions (right) have been explicitly removed from the full mesh leaving behind the interstitial region (left). This partitioning (by element) results in a total of $n_{at} + 1$ different matrices describing the system. The interface nodes between an atomic and interstitial region, however, will exist within both meshes (and matrix systems). These nodes can only be formally defined to exist at one spot within the global solution input to PFEAST and are defined to exist within the interstitial mesh/matrix. The atomic and interstitial regions are then related through a boundary condition, which can be applied at each complex energy pivot (quadrature node) within PFEAST. A detailed description can be found in [24].

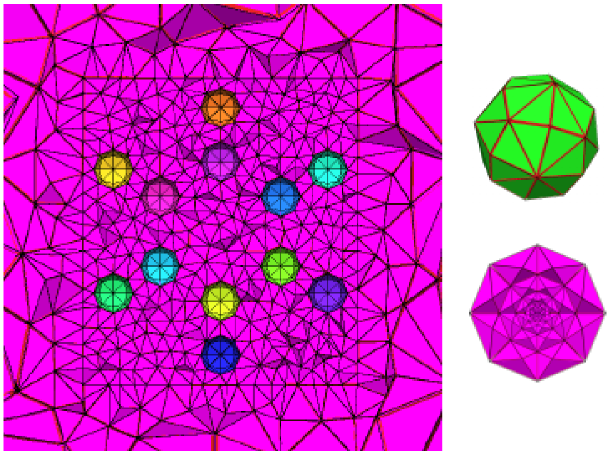

Fig. 3. Example of the muffin-tin decomposition for a benzene molecule ($C_6H_6$). The full mesh is composed of a mesh centered around each atom (right) and the interstitial connecting mesh (left). The interstitial ,mesh has a hole where each atom is located and shares interfaced points with each atomic mesh.

TABLE I
Main Computational Aspects within the Solution Procedure for the Domain-Decomposition MPI Solver. The full matrix (and RHS/solution) is the vector-space summation of the interstitial matrix and all (non-overlapping) atomic matrices.

Solving $HX = Y$ where

$$\{H, X, Y\} = \{H, X, Y\}^{it} \oplus \left( \bigcup_{k=1}^{n_{at}} \{H, X, Y\}_k^{at} \right)$$

1. Obtain boundary condition $\Sigma$ and source term $Z$ in parallel
   Solve: $H_k^{at} X_k^{at} = Y_k^{at} \quad \forall k$
2. Obtain solution in interstitial region
   Solve: $(H^{it} + \Sigma) X^{it} = Y^{it} + Z$
3. Recover solution in atom regions in parallel
   with known boundary values
   Solve: $H_k^{at} X_k^{at} = Y_k^{at} \quad \forall k$

The idea for solving the linear system of equations in (7) is similar to a Schur compliment approach and requires one to obtain, from all atomic system $\{(H_k^{at}, S_k^{at})\}$, a non-local boundary condition (self-energy) $\Sigma_k$ for the interstitial system $(H_{it}, S_{it})$ and a source term $Z_k$ applied to the right-hand-side. After solving the modified equation to obtain the final answer for the interstitial points, the solution within the atomic regions can be recovered by applying Dirichlet boundary conditions at the interface points and solving again a linear system. The three major computational aspects important to the rest of our discussion can be seen in Table I.

We use MKL-PARDISO, a sparse direct solver, for each computational aspect. Solutions for each atomic region (i.e. Steps 1 and 3) can be obtained with perfect parallelism and require the solution to $n_{at}$ moderately sized (independent) sparse

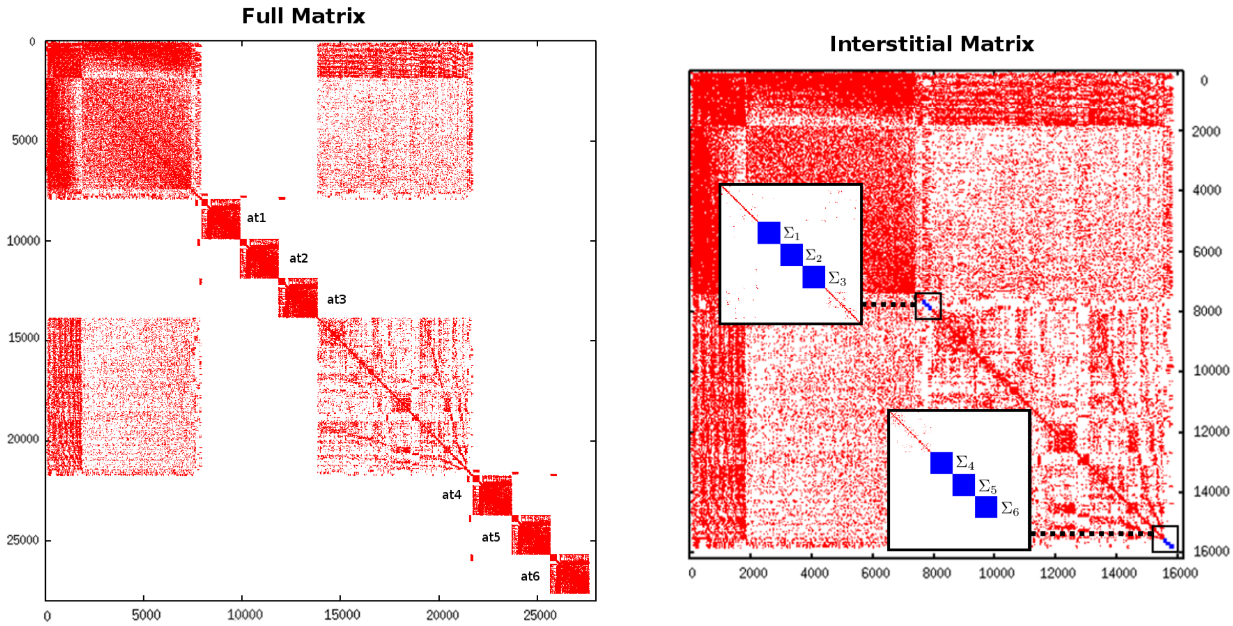

Fig. 4. Structure of a six atom finite-element DFT matrix permuted for 2 **L3** MPI processes (3 atoms per process). The left shows the matrix built from the full mesh including interstitial and atomic regions. The points that correspond to atomic regions are labeled $at_1 - at_6$. The interstitial matrix can be seen on the right, including boundary conditions $\Sigma_1 - \Sigma_6$ (in blue) that account for atomic points. The domain-decomposition approach compresses the large sparse set of atomic point on the left into the 98 × 98 dense blocks $\Sigma$ on the right.

systems. The interstitial matrix (of Step 2), however, will grow linearly with the number of atoms in the molecular system and must be solved in parallel using all **L3** MPI processes and a distributed memory solver. To compute the solution of for interstitial points we use Cluster-MKL-PARDISO.

Interfacing our domain-decomposition solver (DD-Solver) with PFEAST will be slightly more difficult than for Cluster-MKL-PARDISO (acting on the full system) or MUMPS and is similar to what is described in Section IV for the Schur Complement. The solution vector in PFEAST contains points from the interstitial and atomic regions and formally defines the interface nodes to be within the interstitial region. In order to operate on the atomic matrices in Steps 1 and 3 of Table I, the corresponding points must be extracted from the right-hand-side vector supplied by PFEAST. The same is true for Step 2; the distributed piece of the interstitial vector must be extracted for Cluster-MKL-PARDISO. Before returning to the PFEAST kernel, the decomposed solutions must be reconstructed into the representation for PFEAST defined in equation (8), where the $i^{th}$ **L3** MPI process is assigned a collection of rows $x_i$ and $y_i$ of the global vectors. These subsets then contain both atomic and interstitial points. The full interstitial solution

$$X^{it} = \left[\ (x_1^{it})^T,\quad (x_2^{it})^T,\quad \ldots,\quad (x_s^{it})^T\ \right]^T \qquad (16)$$

is divided evenly among the **L3** MPI processes in agreement with Cluster-MKL-PARDISO, with the $i^{th}$ **L3** process assigned a single piece $x_i^{it}$. The atoms are then dealt out like a deck of cards to each **L3** MPI processes, with each owning a collections (or possibly none). The PFEAST vector $q_i$, local to the $i^{th}$ **L3** MPI process, contains the atomic points (with the interface nodes removed) concatenated in order to the end of the local interstitial points.

$$x_i = \left[\ (x_i^{it})^T,\quad (x_{i1}^{at})^T,\quad (x_{i2}^{at})^T,\quad \ldots\ \right]^T. \qquad (17)$$

The data distribution within our finite-element code has been optimized to minimize communication between MPI processes. The interstitial mesh has been permuted so that each local piece that is identified with a given MPI process is guaranteed to include the interface points associated with atoms also assigned to that process. In this way, no communication is needed when applying the boundary conditions.

Figure 4 shows the matrix structure for a (unphysical) six atom carbon ring system permuted specifically for 2 **L3** MPI processes. The plot on the left shows the full matrix including the interstitial and atomic regions. The matrix elements associated with the atoms are the independent square blocks labeled $at_1 - at_6$. The DD-Solver takes in a separate matrix for the interstitial and atomic regions. The interstitial matrix is shown on the right of Figure 4 and contains dense blocks (in blue) that correspond to non-local boundary condition $\Sigma$ and account for effects within each atomic region. The domain-decomposition approach acts to compresses large sparse atomic blocks within the full matrix into the 98 × 98 dense blocks (in blue) within the interstitial matrix. This resulting interstitial system is of much smaller size.

## VI. Results

The matrix systems used to gather results are a collection of Hamiltonians representing carbon nanotube (CNT) molecules of varying length (ranging from 54 to 246 atoms). These finite-element systems use second degree polynomial refinement and are generated from a 3-3 CNT unit cell. They are terminated with six hydrogen atoms at each end and follow a $(U_h|U_aU_b|\ldots|U_aU_b|U_a|U_h)$ pattern (see a 3 unit cell CNT in Figure 5) with $U_h$ representing the six hydrogen atoms and $U_a$ and $U_b$ different configurations of six atom carbon rings. Each carbon atom contains six electrons, which corresponds to three eigenmodes (since the electron spin is not explicitly taken into account). The number of sought eigenvalues $m$ in a CNT system with $n_u$ unit cells is given by

$$m \approx 6 + 18 + 36 \times n_u, \tag{18}$$

where the first six modes correspond to the twelve terminating hydrogen atoms, the next eighteen modes to the $U_a$ carbon ring and the additional factor of thirty-six modes to the $|U_aU_b|$ unit cells.

The experiments were performed on the Mesabi Linux cluster at Minnesota Supercomputing Institute. The compute nodes feature two sockets, each with a twelve core 2.5 GHz Haswell E5-2680v3 processor. We define each MPI process as a single socket and allow the distributed memory solver to use 12 openMP threads per MPI process. The nodes are also equipped with 64 GB of system memory.

Although the PFEAST kernel could be interfaced with an iterative or hybrid solver, we consider only the application of three different sparse direct solvers. Both Cluster-MKL-PARDISO and MUMPS have been tested with the full matrices generated by stitching together the interstitial and atomic meshes within our finite-element electronic structure code (as seen in the left plot of Figure 4). Our application specific domain-decomposition solver (DD-Solver) reorders the matrix based on the number of **L3** MPI processes, in order to reduce communication. For consistency the matrices have been permuted the exact same way for Cluster-MKL-PARDISO, MUMPS and the DD-Solver. All tests have been run with default parameters for the solvers operating in 'in-core' mode. Both Cluster-MKL-PARDISO and MUMPS decompose the full matrix by row over **L3** MPI processes and contain interstitial and atomic points. The inputs to PFEAST will be consistent among the solvers with each **L3** MPI process assigned the exact same subset of the eigenvector solutions, as long as the number of **L3** MPI processes is divisible by the number of atoms.

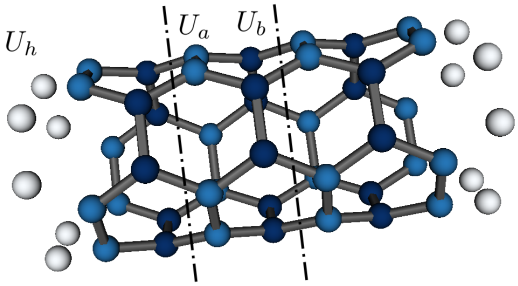

Fig. 5. 3-dimensional plot of a 3 unit cell CNT (3-CNT). The molecule is composed of 3 $|U_aU_b|$ units cells, includes an additional $U_a$ ring, and is terminated with hydrogen atoms $U_h$ at each end.

The scalability of PFEAST will mainly depend on solving the complex linear systems and choosing how to distribute the parallel resources. This results in the trade-off between memory and performance mentioned in Section III. If enough resources are available, it is advisable to place as many MPI processes at the second level of parallelism as possible. This level handles the independent linear systems and requires a copy of the matrix and solution on each **L3** MPI process. It should result in close to ideal linear scaling. In contrast, if the matrix or solution can not fit into memory, it may be distributed using the third level of parallelism. The scalability then depends on the distributed memory solver.

### A. Strong Scalability of L2

The **L2** scalability can be seen in Figure 6, where 3 MPI processes have been assigned to **L3** and the number of clusters of **L3** MPI processes (i.e. the number of **L2**-Communication-Worlds) is varied from 1 to 16 for a total of $3, 6, \ldots, 48$ MPI processes (i.e. a total of 36 to 576 cores). These results have been gathered using the matrix system CNT-5 comprised of five unit-cells with a total of 78 atoms and a dimension of $302,295$. We note that the eigenvalue and eigenvectors obtained for this example correspond the true converged DFT solutions. The right graph in Figure 6 shows the ideal speedup in black that increases linearly with the number of **L2**-Communication-Worlds. The actual speedups (compared to a single **L2**-Communication-World) for the three solvers are plotted as points below the line. We see very close to linear scaling at this level. The total time to solve the eigenvalue problem is plotted on the left. Four PFEAST iterations were needed to reach the default convergence criteria of $10^{-12}$ on the eigenvector residuals. The factorization stage was only computed once for the case of 16 **L2**-Communication-Worlds, but did not result in super linear scaling due the 20 total seconds spent in the PFEAST kernel computing the solution to the reduced eigenvalue problem and performing communication.

### B. Strong Scalability of L3

The scalability of the third level of parallelism **L3** is limited by the properties of the eigensystem and linear-system solver. The strong scalability of the DD-Solver, specifically, may be limited by the number atoms. All **L3** MPI processes can work together in the solution to the interstitial matrix (i.e. Step 2 of Table I) regardless of the number of atoms. But, if the number of **L3** MPI processes exceeds the number of atoms, Steps 1 and 3 in Table I can only be performed on a subset of the MPI processes (some will not have been assigned an atom). It can also be unbalanced if a different number of

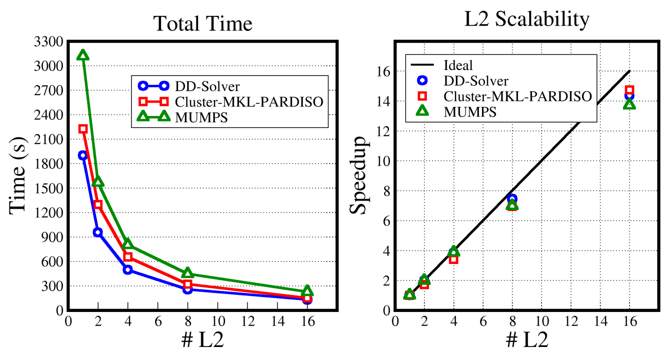

Fig. 6. Scalability of the second level of parallelism. The number of **L2** processes is increased from 1 to 16 while keeping the number of **L3** MPI processes at a constant value of 3 (for a total of 36 to 576 cores). This level is limited to 16 **L2** MPI processes since 16 quadrature points were used. These results are for the CNT-5 eigensystem with $m_0 = 600$ (226 eigenvalues). The left graph gives the total time to solve the eigenvalue problem. The speedup compared to 1 cluster of **L3** processes can be seen on the right.

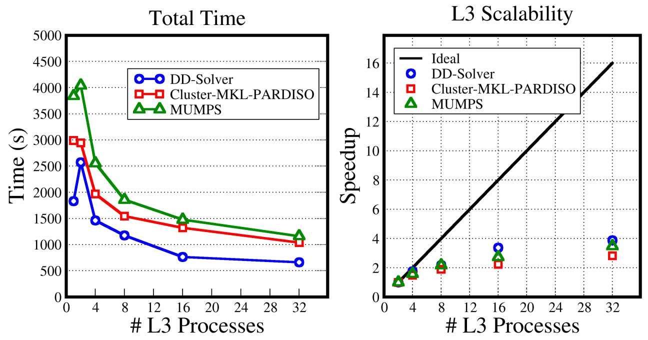

Fig. 7. Scalability of the third level of parallelism for 5-CNT system with $m_0 = 600$ (226 eigenvalues). The left graph the total time to solve the eigenvalue problem. Fully utilizing **L2** (with a total of 12,288 cores) these times could be reduced by a factor of 16. The scaling compared to 2 **L3** MPI processes can be seen on the right.

atoms are assigned to each MPI process. However, problems that require a distributed memory solver will usually target nanostructures with many atoms; usually far outnumbering the parallel resources. Figure 7 shows the strong scalability of the **L3** level of parallelism within PFEAST. The matrix system from the CNT-5 molecule is used again with $m_0 = 600$ and the number of **L3** MPI processes is varied from 1 to 32. We report the total time required to solve the eigenvalue problem (four PFEAST iterations to reach convergence) on the left. The speedup compared to 2 **L3** MPI process can be seen on the right. Here we compare with 2 **L3** MPI processes because of the communication overhead associated with the distributed memory solvers. The time should, ideally, drop by half each time the number of MPI processes is doubled. As can be seen in the graph, the efficiency of the **L3** scaling is much worse than for **L2**.

### C. *Fixed Number of Resources*

Another situation common within the scientific computing community is where parallel resources are limited to a specific number processors. Here the three levels of parallelism within PFEAST can help to optimally utilize all parallel resources and achieve good performance. We only consider the second and third levels of parallelism **L2** and **L3**. The question becomes how to divide the resources among the two levels. If 64 nodes are available on the system and 16 PFEAST contour points are used there are then five ways to distribute the resources across the **L2** and **L3** levels of parallelism. There is no restriction for the number of MPI processes applied to the **L3** level. Each **L2**-Communication-World, however, should have the same number of linear systems for load balancing. With 16 contour nodes, the possible values are $1, 2, 4, 8, 16$. In order to fully utilize the resources we choose the number of **L3** MPI processes as $64, 32, 16, 8, 4$. Table II presents the total time to solve the eigenvalue problem for a fixed number of MPI processes, but different distributions of parallel resources over the **L2** and **L3** levels. As we initially predicted, it is optimal to place as many resources as possible at the **L2** level and the best performance happens for one linear system per **L2**-Communication-World.

TABLE II
TIME (IN SECONDS) TO SOLVE THE 5-CNT EIGENVALUE PROBLEM WITH $m_0 = 600$. THE TOTAL NUMBER OF MPI PROCESSES IS HELD CONSTANT, BUT CAN BE SPLIT BETWEEN THE SECOND AND THIRD LEVEL OF PARALLELISM. MORE MPI PROCESSES AT THE **L3** LEVEL WILL REDUCE THE MEMORY REQUIRED BY EACH MPI PROCESS (I.E. THE NUMBER OF ROWS OF THE EIGENVECTOR AND SYSTEM MATRICES ASSIGNED TO EACH PROCESS).

| # L2 | # L3 | # Rows | DD-Solver | PARDISO | MUMPS |
|---|---|---|---|---|---|
| 1 | 64 | **6260** | 639 | 982 | 923 |
| 2 | 32 | 10553 | 323 | 511 | 643 |
| 4 | 16 | 19139 | 172 | 315 | 332 |
| 8 | 8 | 38277 | 116 | 171 | 194 |
| 16 | 4 | 76554 | **104** | **115** | **125** |

### D. *L1 Scalability*

TABLE III
SCALING OF THE **L1** LEVEL OF PARALLELISM. TOGETHER THE THREE CONTOURS C1, C2 AND C3 - DEFINED BY ($\lambda_{min}$ $\lambda_{max}$) - ARE EQUIVALENT TO THE FULL CONTOUR C0. EACH CONTOUR CAN BE TREATED INDEPENDENTLY (IN PARALLEL) AND CALCULATES ONLY A SUBSET OF EIGENVALUES. SUB-DIVIDING THE CONTOUR RESULTS IN A SPEEDUP FOR THE SOLVE STAGE SINCE THE RESULTING LINEAR-SYSTEMS HAVE FEWER RIGHT-HAND-SIDES ($m_0$). THE FACTORIZATION AND SOLVE TIMES FOR A SINGLE PFEAST ITERATION ARE RECORDED IN THE TABLE. RESULTS USE THE CNT-5 MATRIX SYSTEM WITH A TOTAL OF 13 MPI PROCESSES (156 CORES), ALL PLACED AT THE L3 LEVEL.

| | $\lambda_{min}$ | $\lambda_{max}$ | $m_0$ | # Eig | Fact (s) | Solve (s) |
|---|---|---|---|---|---|---|
| C0 | -500 | -2 | 600 | 273 | 96 | **170** |
| C1 | -500 | -100 | 200 | 66 | 94 | **59** |
| C2 | -100 | -16 | 200 | 92 | 99 | **57** |
| C3 | -16 | -2 | 200 | 115 | 96 | **56** |

With the first level of parallelism **L1**, multiple contours can be solved in parallel to reduce $m_0$ (the number of right-hand-sides and projected eigenproblem dimension). If the reduced eigenvalue problem becomes too large it can not be solved with a traditional dense techniques and the contour must be sub-divided. Also, the **L2** and **L3** will eventually saturate and performance can not be improved placing resources at these levels of parallelism. Multiple contours at the first level **L1** can then be used to increase performance by speeding up the solve stage. For this example we have again used the CNT-5 system with 78 atoms and present two separate cases. The first uses a single contour C0 to calculate all eigenmodes with $m_0 = 600$. The next case sub-divides the entire search interval into three separate pieces C1, C2 and C3 which can be calculated in parallel. This first contour C1 targets the core electronic modes within the system that are well separated from valance states. Two contours C2 and C3 are then used to calculate the valance modes, which are much more densely populated. The total time to solve the eigenvalue problem using the **L1** level of parallelism would then depend on the slowest sub-contour. Because the separation between eigenvalues is small the second and third contours could exhibit slower convergence than C1 and require more PFEAST iterations. However, for the sake of comparison we only present the time of a single PFEAST iteration. Here we report the factorization time, which does not change much between the three contours, and the solve time for a single PFEAST iteration. Because $m_0$ is small, the time spent in the kernel is minimal for this example. Dividing the contour into three segments results in a speedup of around three for the linear system solve.

| # MPI | 9 | 17 | 25 | 33 | 41 |
|---|---|---|---|---|---|
| # Atoms | 54 | 102 | 150 | 198 | 246 |
| System Size | 211,110 | 392,650 | 575,760 | 757,934 | 942,157 |

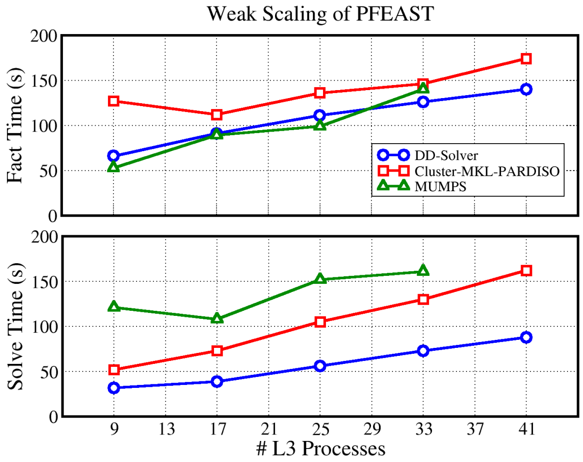

Fig. 8. Weak scaling of factorization and solve stages (in seconds) for a single PFEAST iteration using 16 contour points (16 linear-systems solved in total using # **L2**=1) and 200 right-hand-sides. The matrix size is increased proportionally to the number of MPI processes. Reported timings could be reduced by a factor of 16 if **L2** was fully utilized. MUMPS ran into memory issues for more than 198 atoms (757,934 size, 33 MPI).

### E. *Weak Scalability*

To highlight the weak scalability of PFEAST, we present results for a variety of CNT systems. We consider only the **L3** level of parallelism and place six atoms on each MPI process (each unit cell places an additional 12 atoms in the system). We then gather results, varying the length of the CNT up to 19 unit cells, using $9, 17, \dots, 41$ **L3** MPI processes (up to 492 cores) to optimally utilize the solver. The weak scalability can be seen in Figure 8. Large sparse eigenvalue problems usually attempt to calculate a percentage of the eigenpairs. However, with the **L1** level of parallelism the number of eigenvalues (and right-hand-sides) can be held constant. To showcase the weak scalability we then use a constant value of $m_0 = 200$ for all matrix systems. The number atoms and system size are listed in the table for each molecule. The factorization and solve times for a single iteration of PFEAST are then plotted for all three solvers. We witness, for the factorization, a steeper increase in the time for the smaller systems. The factorization times then begin to taper off for larger systems. The solve time, however continue to increase for larger systems. This behavior is more pronounced for the DD-Solver and Cluster-MKL-PARDISO and is due to communication. MUMPS, however, ran out of system memory far before Cluster-MKL-PARDISO or the DD-Solver. These results show that the DD-Solver is not only faster (in total time), but has better weak scalability than the “black-box” solvers. Much larger molecular systems can then be handled through the domain-decomposition approach.

The total time to solve the eigenvalue problem will increase by an additional multiplication factor depending on the number of PFEAST iterations to reach convergence. However, if the **L2** level was fully utilized, resulting in a maximum of 7,872 cores, the absolute times could be reduced by a factor of 16 as discussed in Figure 6. In electronic structure applications, the number of wanted eigenvalues should also grow linearly with the size of the atomistic system. If the spectrum can be sliced uniformly at level **L1** as illustrated in Table III, and assuming an access to ‘unlimited parallel resources’ where the total number of **L3** MPI processes (and then cores) can grow linearly with the number of slices **L1**, the weak scalability and absolute timing results (that can also be divided by the number of **L2**) presented in Figure 8, shall then be preserved. Solving for all the eigenpairs of the largest system in Figure 6 that contains 246 atoms could use 5 slices each of block size 200, and result in (#**L1**)×(#**L2**)×(#**L3**)=3,280 total MPI processes (using #**L1**=5, #**L2**=16, #**L3**=41) or 3280×12=39,360 total cores (using 12 cores per MPI process).

## VII. Conclusion

This paper outlined the multilevel parallel capabilities of the new PFEAST eigensolver that comprises three full levels of MPI parallelism and operates directly on 1-dimensionally distributed (by row) eigenvector solutions. Due to the multiple levels of parallelism, PFEAST is ideally suited to run on large computing clusters. The three levels of parallelism can work together to minimize time spent in all stages of the algorithm. The third level can be used to reduce the memory per node and to decrease the solution time of both the linear system factorization and solve. The second level has close to ideal scaling and, if fully utilized, can reduce the algorithmic complexity to solving a single complex linear system per PFEAST iteration. Finally, the first level can be used to reduce the memory requirements for storing eigenvector solutions and speed up the solve stage of the linear system solution. Additionally it allows for the computation of a very large number of eigenvalues by subdividing the full search interval. The software is matrix-format independent and can be easily interfaced with any distributed memory solver. Three different solvers, including a custom domain-decomposition solver, have been interfaced with the software kernel and highlight its generality. This, along with its multilevel scalability, will make PFEAST a valuable new tool for the HPC community [29].

## Acknowledgment

This work was supported by NSF Grant Nos. CCF #1510010 and CCF #1505970. We would also like to thank Dr. Ping Tak Peter Tang for many helpful discussions.

## References

[1] P. R. Amestoy, I. S. Duff, J. L'Excellent, and J. Koster, MUMPS*: a general purpose distributed memory sparse solver*, in Applied Parallel Computing. New Paradigms for HPC in Industry and Academia, Springer, 2000, pp. 121–130.

[2] E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammerling, A. McKenney, et al., *LAPACK Users' guide*, vol. 9, SIAM, 1999.

[3] J. Asakura, T. Sakurai, T. Tadano, H., Ikegami, and K. Kimura, *A numerical method for nonlinear eigenvalue problems using contour integrals*, JSIAM Letters, 1 (2009), pp. 52–55.

[4] A. P. Austin and L. N. Trefethen, *Computing eigenvalues of real symmetric matrices with rational filters in real arithmetic*, SIAM Journal on Scientific Computing, 37 (2015), pp. A1365–A1387.

[5] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, et al., *Sca*LAPACK *users' guide*, vol. 4, Siam, 1997.

[6] M. Bollhoefer and Y. Notay, *JADAMILU: a software code for computing selected eigenvalues of large sparse symmetric matrices*, Comput. Phys. Comm., 177 (2007), pp. 951–964.

[7] Z. Chen and E. Polizzi, *Spectral-based propagation schemes for time-dependent quantum systems with application to carbon nanotubes*, Physical Review B, 82 (2010), p. 205410.

[8] E. Di Napoli, E. Polizzi, and Y. Saad, *Efficient estimation of eigenvalue counts in an interval*, Numerical Linear Algebra with Applications, (2016).

[9] FEAST E*igensolver*, 2009–2015. http://www.feast-solver.org/.

[10] B. Gavin and E. Polizzi, *Non-linear eigensolver-based alternative to traditional scf methods*, The Journal of chemical physics, 138 (2013), p. 194101.

[11] B. Gavin and E. Polizzi, *Enhancing the performance and robustness of the feast eigensolver*, arXiv preprint arXiv:1605.08771, (2016).

[12] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, *A high-performance, portable implementation of the mpi message passing interface standard*, Parallel computing, 22 (1996), pp. 789–828.

[13] S. Güttel, E. Polizzi, P. T. P. Tang, and G. Viaud, *Zolotarev quadrature rules and load balancing for the* FEAST *eigensolver*, to appear in SIAM Journal on Scientific Computing, (2015). arXiv preprint arXiv:1407.8078 (2014).

[14] T. Ikegami, T. Sakurai, and U. Nagashima, *A filter diagonalization for generalized eigenvalue problems based on the* S*akurai–*S*ugiura projection method*, Journal of Computational and Applied Mathematics, 233 (2010), pp. 1927–1936.

[15] A. Imakura, L. Du, and T. Sakurai, *A block* A*rnoldi-type contour integral spectral projection method for solving generalized eigenvalue problems*, Applied Mathematics Letters, 32 (2014), pp. 22–27.

[16] I*ntel* M*ath* K*ernel* L*ibrary*. http://software.intel.com/en-us/intel-mkl.

[17] A. Kalinkin and K. Arturov, *Asynchronous approach to memory management in sparse multifrontal methods on multiprocessors*, Applied Mathematics, 2013 (2013).

[18] J. Kestyn, E. Polizzi, and P. T. P. Tang, FEAST *eigensolver for non-hermitian problems*, arXiv preprint arXiv:1506.04463, (2015).

[19] A. V. Knyazev, M. E. Argentati, I. Lashuk, and E. E. Ovtchinnikov, *Block locally optimal preconditioned eigenvalue xolvers (*BLOPEX*) in* HYPRE *and* PETS*c*, SIAM Journal on Scientific Computing, 29 (2007), pp. 2224–2239.

[20] W. Kohn and L. J. Sham, *Self-consistent equations including exchange and correlation effects*, Physical review, 140 (1965), p. A1133.

[21] S. E. Laux, *Solving complex band structure problems with the* FEAST *eigenvalue algorithm*, Physical Review B, 86 (2012), p. 075103.

[22] R. B. Lehoucq, D. C. Sorensen, and C. Yang, ARPACK *Users' guide: solution of large-scale eigenvalue problems with implicitly restarted* A*rnoldi methods*, vol. 6, SIAM, 1998.

[23] L. Lehtovaara, V. Havu, and M. Puska, *All-electron density functional theory and time-dependent density functional theory with high-order finite elements*, The Journal of chemical physics, 131 (2009), p. 054103.

[24] A. R. Levin, D. Zhang, and E. Polizzi, FEAST *fundamental framework for electronic structure calculations: Reformulation and solution of the muffin-tin problem*, Computer Physics Communications, 183 (2012), pp. 2370–2375.

[25] K. Mendiratta and E. Polizzi, *A threaded spike algorithm for solving general banded systems*, Parallel Computing, 37 (2011), pp. 733–741.

[26] B. A. Parlett, *The symmetric eigenvalue problem*, vol. 7, SIAM, 1980.

[27] E. Polizzi, *Density-matrix-based algorithm for solving eigenvalue problems*, Physical Review B, 79 (2009), p. 115112.

[28] E. Polizzi and J. Kestyn, *A high-performance numerical library for solving eigenvalue problems:* FEAST *solver v3.0 user's guide*, arXiv preprint arXiv:1203.4031, (2015).

[29] ———, FEAST*v4.0 update*, Release in Preparation, (2016).

[30] E. Polizzi and A. H. Sameh, *A parallel hybrid banded system solver: the spike algorithm*, Parallel computing, 32 (2006), pp. 177–194.

[31] Y. Saad, *Numerical methods for large eigenvalue problems*, vol. 158, SIAM, 1992.

[32] T. Sakurai and H. Sugiura, *A projection method for generalized eigenvalue problems using numerical integration*, Journal of Computational and Applied Mathematics, 159 (2003), pp. 119–128.

[33] T. Sakurai and H. Tadano, CIRR*: a* R*ayleigh-*R*itz type method with contour integral for generalized eigenvalue problems*, Hokkaido Mathematical Journal, 36 (2007), pp. 745–757.

[34] O. Schenk and K. Gärtner, *On fast factorization pivoting methods for sparse symmetric indefinite systems*, Electronic Transactions on Numerical Analysis, 23 (2006), pp. 158–179.

[35] B. Spring, *Enhanced capabilities of the spike algorithm and a new spike-openmp solver*, Master's thesis, University of Massachusetts Amherst, 2014.

[36] A. Stathopoulos and J. R. McCombs, PRIMME*: Preconditioned* I*terative* M*ultimethod* E*igensolver: methods and software description*, ACM Transactions on Mathematical Software, 37 (2010), p. 21.

[37] P. T. P. Tang, J. Kestyn, and E. Polizzi, *A new highly parallel non-*H*ermitian eigensolver*, in Proceedings of the High Performance Computing Symposium, San Diego, CA, USA, 2014, Society for Computer Simulation International.

[38] P. T. P. Tang and E. Polizzi, FEAST *as a subspace iteration eigensolver accelerated by approximate spectral projection*, SIAM Journal on Matrix Analysis and Applications, 35 (2014), pp. 354–390.

[39] A. Toselli and O. B. Widlund, *Domain decomposition methods: algorithms and theory*, vol. 34, Springer, 2005.

[40] G. Yin, R. H. Chan, and M. Yeung, *A* FEAST *algorithm for generalized non-*H*ermitian eigenvalue problems*, arXiv preprint arXiv:1404.1768, (2014).