Li and Ying *Res Math Sci(2017)4:12*
DOI 10.1186/s40687-017-0100-6

Research in the Mathematical Sciences

RESEARCH Open Access

Distributed-memory hierarchical interpolative factorization

Yingzhou Li[1] and Lexing Ying[1,2*]

*Correspondence:
lexing@stanford.edu
[2]Department of Mathematics, Stanford University, Stanford, CA, USA
Full list of author information is available at the end of the article

Abstract

The hierarchical interpolative factorization (HIF) offers an efficient way for solving or preconditioning elliptic partial differential equations. By exploiting locality and low-rank properties of the operators, the HIF achieves quasi-linear complexity for factorizing the discrete positive definite elliptic operator and linear complexity for solving the associated linear system. In this paper, the distributed-memory HIF (DHIF) is introduced as a parallel and distributed-memory implementation of the HIF. The DHIF organizes the processes in a hierarchical structure and keeps the communication as local as possible. The computation complexity is $O(\frac{N\log N}{P})$ and $O(\frac{N}{P})$ for constructing and applying the DHIF, respectively, where N is the size of the problem and P is the number of processes. The communication complexity is $O(\sqrt{P}\log^3 P)\alpha + O(\frac{N^{2/3}}{\sqrt{P}})\beta$ where α is the latency and β is the inverse bandwidth. Extensive numerical examples are performed on the NERSC Edison system with up to 8192 processes. The numerical results agree with the complexity analysis and demonstrate the efficiency and scalability of the DHIF.

Keywords: Sparse matrix, Multifrontal, Elliptic problem, Matrix factorization, Structured matrix

Mathematics Subject Classification: 44A55, 65R10, 65T50

1 Background

This paper proposes an efficient distributed-memory algorithm for solving elliptic partial differential equations (PDEs) of the form,

$$-\nabla\cdot(a(x)\nabla u(x)) + b(x)u(x) = f(x), \quad x\in\Omega\subset\mathbb{R}^3, \tag{1}$$

with a certain boundary condition, where $a(x) > 0$, $b(x)$ and $f(x)$ are given functions and $u(x)$ is an unknown function. Since this elliptic equation is of fundamental importance to problems in physical sciences, solving (1) effectively has a significant impact in practice. Discretizing this with local schemes such as the finite difference or finite element methods leads to a sparse linear system,

$$Au = f, \tag{2}$$

where $A\in\mathbb{R}^{N\times N}$ is a sparse symmetric matrix with $O(N)$ nonzero entries with N being the number of the discretization points, and u and f are the discrete approximations of the functions $u(x)$ and $f(x)$, respectively. For many practical applications, one often

Springer Open

needs to solve (1) on a sufficient fine mesh for which N can be very large, especially for three-dimensional (3D) problems. Hence, there is a practical need for developing fast and parallel algorithms for the efficient solution of (1).

1.1 Previous work

A great deal of effort in the field of scientific computing has been devoted to the efficient solution of (2). Beyond the $O(N^3)$ complexity naïve matrix inversion approach, one can classify the existing fast algorithms into the following groups.

The first one consists of the sparse direct algorithms, which take advantage of the sparsity of the discrete problem. The most noticeable example in this group is the nested dissection multifrontal method (MF) method [14,16,26]. By carefully exploring the sparsity and the locality of the problem, the multifrontal method factorizes the matrix A (and thus A^{-1}) as a product of sparse lower and upper triangular matrices. For 3D problems, the factorization step costs $O(N^2)$ operations, while the application step takes $O(N^{4/3})$ operations. Many parallel implementations [3,4,30] of the multifrontal method were proposed and they typically work quite well for problem of moderate size. However, as the problem size goes beyond a couple of millions, most implementations, including the distributed-memory ones, hit severe bottlenecks in memory consumption.

The second group consists of iterative solvers [9,15,33,34], including famous algorithms such as the conjugate gradient (CG) method and the multigrid method. Each iteration of these algorithms typically takes $O(N)$ steps and hence the overall cost for solving (2) is proportional to the number of iterations required for convergence. For problems with smooth coefficient functions $a(x)$ and $b(x)$, the number of iterations typically remains rather small and the optimal linear complexity is achieved. However, if the coefficient functions lack regularity or have high contrast, the iteration number typically grows quite rapidly as the problem size increases.

The third group contains the methods based on structured matrices [6–8,11]. These methods, for example, the $\mathcal{H}$-matrix [18,20], the $\mathcal{H}^2$-matrix [19], and the hierarchically semi-separable matrix (HSS) [10,42], are shown to have efficient approximations of linear or quasi-linear complexity for the matrices A and A^{-1}. As a result, the algebraic operations of these matrices are of linear or quasi-linear complexities as well. More specifically, the recursive inversion and the rocket-style inversion [1] are two popular methods for the inverse operation. For distributed-memory implementations, however, the former lacks parallel scalability [24,25], while the latter demonstrates scalability only for 1D and 2D problems [1]. For 3D problems, these methods typically suffer from large prefactors that make them less efficient for practical large-scale problems.

A recent group of methods explores the idea of integrating the MF method with the hierarchical matrix [17,21,28,32,38–41] or block low-rank matrix [2,35,36] approach in order to leverage the efficiency of both methods. Instead of directly applying the hierarchical matrix structure to the 3D problems, these methods apply it to the representation of the frontal matrices (i.e., the interactions between the lower-dimensional fronts). These methods are of linear or quasi-linear complexities in theory with much small prefactors. However, due to the combined complexity, the implementation is highly non-trivial and quite difficult for parallelization [27,43].

More recently, the hierarchical interpolative factorization (HIF) [22,23] is proposed as a new way for solving elliptic PDEs and integral equations. As compared to the multifrontal method, the HIF includes an extra step of skeletonizing the fronts in order to reduce the size of the dense frontal matrices. Based on the key observation that the number of skeleton points on each front scales linearly as the one-dimensional fronts, the HIF factorizes the matrix A (and thus A^{-1}) as a product of sparse matrices that contains only $O(N)$ nonzero entries in total. In addition, the factorization and application of the HIF are of complexities $O(N \log N)$ and $O(N)$, respectively, for N being the total number of degrees of freedom (DOFs) in (2). In practice, the HIF shows significant saving in terms of computational resources required for 3D problems.

1.2 Contribution

This paper proposes the first *distributed-memory* hierarchical interpolative factorization (DHIF) for solving very large-scale problems. In a nutshell, the DHIF organizes the processes in an octree structure in the same way that the HIF partitions the computation domain. In the simplest setting, each leaf node of the computation domain is assigned a single process. Thanks to the locality of the operator in (1), this process only communicates with its neighbors and all algebraic computations are local within the leaf node. At higher levels, each node of the computation domain is associated with a process group that contains all processes in the subtree starting from this node. The computations are all local within this process group via parallel dense linear algebra, and the communications are carried out between neighboring process groups. By following this octree structure, we make sure that both the communication and computations in the distributed-memory HIF are evenly distributed. As a result, the distributed-memory HIF implementation achieves $O(\frac{N \log N}{P})$ and $O(\frac{N}{P})$ parallel complexity for constructing and applying the factorization, respectively, where N is the number of DOFs and P is the number of processes.

We have performed extensive numerical tests. The numerical results support the complexity analysis of the distributed-memory HIF and suggest that the DHIF is a scalable method up to thousands of processes and can be applied to solve large-scale elliptic PDEs.

1.3 Organization

The rest of this paper is organized as follows. In Sect. 2, we review the basic tools needed for both HIF and DHIF, and review the sequential HIF. Section 3 presents the DHIF as a parallel extension of the sequential HIF for 3D problems. Complexity analyses for memory usage, computation time, and communication volume are given at the end of this section. The numerical results detailed in Sect. 4 show that the DHIF is applicable to large-scale problems and achieves parallel scalability up to thousands of processes. Finally, Sect. 5 concludes with some extra discussions on future work.

2 Preliminaries

This section reviews the basic tools and the sequential HIF. First, we start by listing the notations that are widely used throughout this paper.

2.1 Notations

In this paper, we adopt MATLAB notations for simple representation of submatrices. For example, given a matrix A and two index sets, s_1 and s_2, $A(s_1, s_2)$ represents the submatrix of A with the row indices in s_1 and column indices in s_2. The next two examples explore the

usage of MATLAB notation ":." With the same settings, $A(s_1, :)$ represents the submatrix of A with row indices in s_1 and all columns. Another usage of notation ":" is to create regularly spaced vectors for integer values i and j, for instance, $i : j$ is the same as $[i, i+1, i+2, \ldots, j]$ for $i \leq j$.

In order to simplify the presentation, we consider the problem (1) with periodic boundary condition and assume that the domain $\Omega = [0, 1)^3$ and is discretized with a grid of size $n \times n \times n$ for $n = 2^L m$, where $L = O(\log n)$ and $m = O(1)$ are both integers. In the rest of this paper, $L + 1$ is known as the number of levels in the hierarchical structure and L is the level number of the root level. We use $N = n^3$ to denote the total number of DOFs, which is the dimension of the sparse matrix A in (2). Furthermore, each grid point $\mathbf{x_j}$ is defined as

$$\mathbf{x_j} = h\mathbf{j} = h(j_1, j_2, j_3), \tag{3}$$

where $h = 1/n$, $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \leq j_1, j_2, j_3 < n$.

In order to fully explore the hierarchical structure of the problem, we recursively bipartite each dimension of the grid into $L + 1$ levels. Let the leaf level be level 0 and the root level be level L. At level ℓ, a cell indexed with $\mathbf{j}$ is of size $m2^\ell \times m2^\ell \times m2^\ell$ and each point in the cell is in the range, $\left[m2^\ell j_1 + (0 : m2^\ell - 1)\right] \times \left[m2^\ell j_2 + (0 : m2^\ell - 1)\right] \times \left[m2^\ell j_3 + (0 : m2^\ell - 1)\right]$, for $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \leq j_1, j_2, j_3 < 2^{L-\ell}$. $C_{\mathbf{j}}^\ell$ denotes the grid point set of the cell at level ℓ indexed with $\mathbf{j}$.

A cell $C_{\mathbf{j}}^\ell$ owns three faces: top, front, and left. Each of these three faces contains the grid points on the first frame in the corresponding direction. For example, the front face contains the grid points in $\left[m2^\ell j_1 + (0 : m2^\ell - 1)\right] \times \left[m2^\ell j_2\right] \times \left[m2^\ell j_3 + (0 : m2^\ell - 1)\right]$. Besides these three in-cell faces (top, front, and left) that are owned by a cell, each cell is also adjacent to three out-of-cell faces (bottom, back, right) owned by its neighbors. Each of these three faces contains the grid points on the next to the last frame in the corresponding dimension. As a result, these faces contain DOFs that belong to adjacent cells. For example, the bottom face of $C_{\mathbf{j}}^\ell$ contains the grid points in $\left[m2^\ell(j_1 + 1)\right] \times \left[m2^\ell j_2 + (0 : m2^\ell - 1)\right] \times \left[m2^\ell j_3 + (0 : m2^\ell - 1)\right]$. These six faces are the surrounding faces of $C_{\mathbf{j}}^\ell$. One also defines the interior of $C_{\mathbf{j}}^\ell$ to be $I_{\mathbf{j}}^\ell = \left[m2^\ell j_1 + (1 : m2^\ell - 1)\right] \times \left[m2^\ell j_2 + (1 : m2^\ell - 1)\right] \times \left[m2^\ell j_3 + (1 : m2^\ell - 1)\right]$ for the same $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \leq j_1, j_2, j_3 < 2^{L-\ell}$. Figure 1 shows an illustration of a cell, its faces, and its interior. These definitions and notations are summarized in Table 1. Also included here are some notations used for the processes, which will be introduced later.

2.2 Sparse elimination

Suppose that A is a symmetric matrix. The row/column indices of A are partitioned into three sets $I \bigcup F \bigcup R$ where I refers to the interior point set, F refers to the surrounding face point set, and R refers to the rest point set. We further assume that there is no interaction between the indices in I and the ones in R. As a result, one can write A in the following form

$$A = \begin{bmatrix} A_{II} & A_{FI}^T & \\ A_{FI} & A_{FF} & A_{RF}^T \\ & A_{RF} & A_{RR} \end{bmatrix}. \tag{4}$$

Let the LDL^T decomposition of A_{II} be $A_{II} = L_I D_I L_I^T$, where L_I is lower triangular matrix with unit diagonal. According to the block Gaussian elimination of A given by (4),

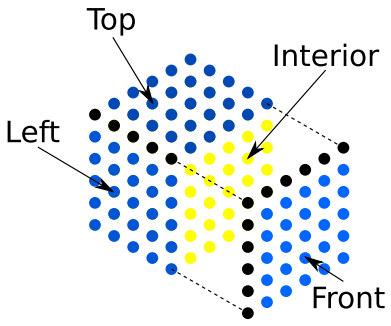

Fig. 1 Cell structure: *top*, *front*, *left*, and *interior* points are indicated by *arrows*; bottom, back, and right points are not plotted in the figure; the *black dots* denote the edge points; the *dash line* indicates that the front frame is pulled away in order to show the interior points

Table 1 Commonly used notations

Notation	Description
n	Number of points in each dimension of the grid
N	Number of points in the grid
h	Grid gap size
ℓ	Level number in the hierarchical structure
L	Level number of the root level in the hierarchical structure
$\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$	Unit vector along each dimension
$\mathbf{0}$	Zero vector
$\mathbf{j}$	Triplet index $\mathbf{j} = (j_1, j_2, j_3)$
$\mathbf{x}_\mathbf{j}$	Point on the grid indexed with $\mathbf{j}$
Ω	The set of all points on the grid
$C_\mathbf{j}^\ell$	Cell at level ℓ with index $\mathbf{j}$
$\mathcal{C}^\ell$	$\mathcal{C}^\ell = \{C_\mathbf{j}^\ell\}_\mathbf{j}$ is the set of all cells at level ℓ
$\mathcal{F}_\mathbf{j}^\ell$	Set of all surrounding faces of cell $C_\mathbf{j}^\ell$
$\mathcal{F}^\ell$	Set of all faces at level ℓ
$I_\mathbf{j}^\ell$	Interior of $C_\mathbf{j}^\ell$
$\mathcal{I}^\ell$	$\mathcal{I}^\ell = \{I_\mathbf{j}^\ell\}_\mathbf{j}$ is the set of all interiors at level ℓ
Σ^ℓ	The set of active DOFs at level ℓ
$\Sigma_\mathbf{j}^\ell$	The set of active DOFs at level ℓ with process group index $\mathbf{j}$
$p_\mathbf{j}^\ell, p^\ell$	The process group at level ℓ with/without index $\mathbf{j}$

one defines the *sparse elimination* to be

$$S_I^T A S_I = \begin{bmatrix} D_I & & \\ & B_{FF} & A_{RF}^T \\ & A_{RF} & A_{RR} \end{bmatrix}, \tag{5}$$

where $B_{FF} = A_{FF} - A_{FI}A_{II}^{-1}A_{FI}^T$ is the associated Schur complement and the explicit expressions for S_I is

$$S_I = \begin{bmatrix} L_I^{-T} & -A_{II}^{-1}A_{FI}^T & \\ & I & \\ & & I \end{bmatrix}. \tag{6}$$

The sparse elimination removes the interaction between the interior points I and the corresponding surrounding face points F and leaves A_{RF} and A_{RR} untouched. We call the entire point set, $I \bigcup F \bigcup R$, the *active point set*. Then, after the sparse elimination,

the interior points are decoupled from other points, which is conceptually equivalent to eliminate the interior points from the active point set. After this, the new active point set can be regarded as $F \bigcup R$.

Figure 2 illustrates the impact of the sparse elimination. The dots in the figure represent the active points. Before the sparse elimination (left), edge points, face points, and interior points are active, while after the sparse elimination (right) the interior points are eliminated from the active point set.

2.3 Skeletonization

Skeletonization is a tool for eliminating redundant point set from a symmetric matrix that has low-rank off-diagonal blocks. The key step in skeletonization uses the interpolative decomposition [12,29] of low-rank matrices.

Let A be a symmetric matrix of the form,

$$A = \begin{bmatrix} A_{FF} & A_{RF}^T \\ A_{RF} & A_{RR} \end{bmatrix}, \tag{7}$$

where A_{RF} is a numerically low-rank matrix. The interpolative decomposition of A_{RF} is (up to a permutation)

$$A_{RF} = \begin{bmatrix} A_{R\overline{\widehat{F}}} & A_{R\widehat{F}} \end{bmatrix} \approx \begin{bmatrix} A_{R\widehat{F}} T_F & A_{R\widehat{F}} \end{bmatrix}, \tag{8}$$

where T_F is the interpolation matrix, $\widehat{F}$ is the skeleton point set, $\overline{\widehat{F}}$ is the redundant point set, and $F = \widehat{F} \bigcup \overline{\widehat{F}}$. Applying this approximation to A results

$$A \approx \left[\begin{array}{cc|c} A_{\overline{\widehat{F}}\,\overline{\widehat{F}}} & A_{\widehat{F}\overline{\widehat{F}}}^T & T_F^T A_{R\widehat{F}}^T \\ A_{\widehat{F}\overline{\widehat{F}}} & A_{\widehat{F}\widehat{F}} & A_{R\widehat{F}}^T \\ \hline A_{R\widehat{F}} T_F & A_{R\widehat{F}} & A_{RR} \end{array}\right], \tag{9}$$

and be symmetrically factorized as

$$S_{\overline{\widehat{F}}}^T Q_F^T A Q_F S_{\overline{\widehat{F}}} \approx S_{\overline{\widehat{F}}}^T \left[\begin{array}{cc|c} B_{\overline{\widehat{F}}\,\overline{\widehat{F}}} & B_{\widehat{F}\overline{\widehat{F}}}^T & \\ B_{\widehat{F}\overline{\widehat{F}}} & A_{\widehat{F}\widehat{F}} & A_{R\widehat{F}}^T \\ \hline & A_{R\widehat{F}} & A_{RR} \end{array}\right] S_{\overline{\widehat{F}}} = \left[\begin{array}{cc|c} D_{\overline{\widehat{F}}} & & \\ & B_{\widehat{F}\widehat{F}} & A_{R\widehat{F}}^T \\ \hline & A_{R\widehat{F}} & A_{RR} \end{array}\right], \tag{10}$$

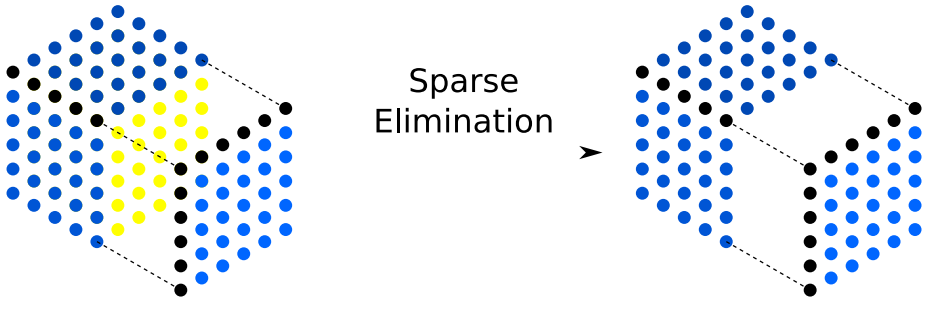

Fig. 2 Sparse elimination: the interior points are eliminated after the sparse elimination; the rest points are not all plotted in the figure

where

$$B_{\overline{\overline{F}}\,\overline{\overline{F}}} = A_{\overline{\overline{F}}\,\overline{\overline{F}}} - T_F^T A_{\widehat{F}\overline{\overline{F}}} - A_{\widehat{F}\overline{\overline{F}}}^T T_F + T_F^T A_{\widehat{F}\widehat{F}} T_F, \tag{11}$$

$$B_{\widehat{F}\overline{\overline{F}}} = A_{\widehat{F}\overline{\overline{F}}} - A_{\widehat{F}\widehat{F}} T_F, \tag{12}$$

$$B_{\widehat{F}\widehat{F}} = A_{\widehat{F}\widehat{F}} - B_{\widehat{F}\overline{\overline{F}}} B_{\overline{\overline{F}}\,\overline{\overline{F}}}^{-1} B_{\widehat{F}\overline{\overline{F}}}^T. \tag{13}$$

The factor Q_F is generated by the block Gaussian elimination, which is defined to be

$$Q_F = \left[\begin{array}{cc|c} I & & \\ -T_F & I & \\ \hline & & I \end{array}\right]. \tag{14}$$

Meanwhile, the factor $S_{\overline{\overline{F}}}$ is introduced in the sparse elimination:

$$S_{\overline{\overline{F}}} = \left[\begin{array}{cc|c} L_{\overline{\overline{F}}}^{-T} & -B_{\overline{\overline{F}}\,\overline{\overline{F}}}^{-1} B_{\widehat{F}\overline{\overline{F}}}^T & \\ & I & \\ \hline & & I \end{array}\right] \tag{15}$$

where $L_{\overline{\overline{F}}}$ and $D_{\overline{\overline{F}}}$ come from the LDL^T factorization of $B_{\overline{\overline{F}}\,\overline{\overline{F}}}$, i.e., $B_{\overline{\overline{F}}\,\overline{\overline{F}}} = L_{\overline{\overline{F}}} D_{\overline{\overline{F}}} L_{\overline{\overline{F}}}^T$. Similar to what happens in Sect. 2.2, the skeletonization eliminates the redundant point set $\overline{\overline{F}}$ from the active point set.

The point elimination idea of the skeletonization is illustrated in Fig. 3. Before the skeletonization (left), the edge points, interior points, skeleton face points (red), and redundant face points (pink) are all active, while after the skeletonization (right) the redundant face points are eliminated from the active point set.

2.4 Sequential HIF

This section reviews the sequential hierarchical interpolative factorization (HIF) for 3D elliptic problems (1) with the periodic boundary condition. Without loss of generality, we discretize (1) with the seven-point stencil on a uniform grid, which is defined in Sect. 2.1. The discrete system is

$$\begin{aligned} &\frac{1}{h^2}\left(a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_1} + a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_1} + a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_2} + a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_2} + a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_3} + a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_3}\right) u_{\mathbf{j}} \\ &-\frac{1}{h^2}\left(a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_1} u_{\mathbf{j}-\mathbf{e}_1} + a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_1} u_{\mathbf{j}+\mathbf{e}_1} + a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_2} u_{\mathbf{j}-\mathbf{e}_2}\right. \\ &\left. + a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_2} u_{\mathbf{j}+\mathbf{e}_2} + a_{\mathbf{j}-\frac{1}{2}\mathbf{e}_3} u_{\mathbf{j}-\mathbf{e}_3} + a_{\mathbf{j}+\frac{1}{2}\mathbf{e}_3} u_{\mathbf{j}+\mathbf{e}_3}\right) + b_{\mathbf{j}} u_{\mathbf{j}} = f_{\mathbf{j}} \end{aligned} \tag{16}$$

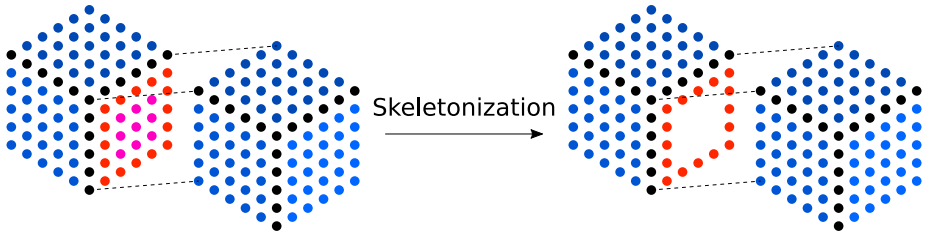

Fig. 3 Skeletonization: the working face is colored by *red* and *pink*; *red points* are the skeleton points on the face, whereas *pink points* are the redundant points on the face; skeletonization eliminates the redundant points from the active point set

at each grid point $\mathbf{x_j}$ for $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \le j_1, j_2, j_3 < n$, where $a_\mathbf{j} = a(\mathbf{x_j})$, $b_\mathbf{j} = b(\mathbf{x_j})$, $f_\mathbf{j} = f(\mathbf{x_j})$, and $u_\mathbf{j}$ approximates the unknown function $u(x)$ at $\mathbf{x_j}$. The corresponding linear system is

$$Au = f \tag{17}$$

where A is a sparse SPD matrix if $b > 0$.

We first introduce the notion of active and inactive DOFs.

- A set Σ of DOFs of A are called **active** if $A_{\Sigma\Sigma}$ is not a diagonal matrix or $A_{\bar{\Sigma}\Sigma}$ is a nonzero matrix;
- A set Σ of DOFs of A are called **inactive** if $A_{\Sigma\Sigma}$ is a diagonal matrix and $A_{\bar{\Sigma}\Sigma}$ is a zero matrix.

Here $\bar{\Sigma}$ refers to the complement of the set Σ. Sparse elimination and skeletonization provide concrete examples of active and inactive DOFs. For example, sparse elimination turns the indices I from active DOFs of A to inactive DOFs of $\widetilde{A} = S_I^T A S_I$ in (5). Skeletonization turns the indices $\overline{\overline{F}}$ from active DOFs of A to inactive DOFs of $\widetilde{A} = S_{\overline{\overline{F}}}^T Q_F^T A Q_F S_{\overline{\overline{F}}}$ in (10).

With these notations, the sequential HIF in [22] is summarized as follows. A more illustrative representation of the sequential HIF is given on the left column of Fig. 5.

- *Preliminary* Let $A^0 = A$ be the sparse symmetric matrix in (17), Σ^0 be the initial active DOFs of A, which includes all indices.
- *Level ℓ for $\ell = 0, \ldots, L-1$.*

  – *Preliminary* Let A^ℓ denote the matrix before any elimination at level ℓ. Σ^ℓ is the corresponding active DOFs. Let us recall the notations in Sect. 2.1. $C_\mathbf{j}^\ell$ denotes the active DOFs in the cell at level ℓ indexed with $\mathbf{j}$. $\mathcal{F}_\mathbf{j}^\ell$ and $I_\mathbf{j}^\ell$ denote the surrounding faces and interior active DOFs in the corresponding cell, respectively.

  – *Sparse elimination* We first focus on a single cell at level ℓ indexed with $\mathbf{j}$, i.e., $C_\mathbf{j}^\ell$. To simplify the notation, we drop the superscript and subscript for now and introduce $C = C_\mathbf{j}^\ell$, $I = I_\mathbf{j}^\ell$, $F = \mathcal{F}_\mathbf{j}^\ell$, and $R = R_\mathbf{j}^\ell$. Based on the discretization and previous level eliminations, the interior active DOFs interact only with itself and its surrounding faces. The interactions of the interior active DOFs and the rest DOFs are empty and the corresponding matrix is zero, $A^\ell(R, I) = 0$. Hence, by applying sparse elimination, we have,

$$S_I^T A^\ell S_I = \begin{bmatrix} D_I & & \\ & B_{FF}^\ell & \left(A_{RF}^\ell\right)^T \\ & A_{RF}^\ell & A_{RR}^\ell \end{bmatrix}, \tag{18}$$

  where the explicit definitions of B_{FF}^ℓ and S_I are given in the discussion of sparse elimination. This factorization eliminates I from the active DOFs of A^ℓ.

  Looping over all cells $C_\mathbf{j}^\ell$ at level ℓ, we obtain

$$\widetilde{A}^\ell = \left(\prod_{I \in \mathcal{I}^\ell} S_I\right)^T A^\ell \left(\prod_{I \in \mathcal{I}^\ell} S_I\right), \tag{19}$$

$$\widetilde{\Sigma}^\ell = \Sigma^\ell \setminus \bigcup_{I \in \mathcal{I}^\ell} I. \tag{20}$$

Now all the active interior DOFs at level ℓ are eliminated from Σ^ℓ.

– *Skeletonization* Each face at level ℓ not only interacts within its own cell but also interacts with faces of neighbor cells. Since the interaction between any two different faces is low rank, this leads us to apply skeletonization. The skeletonization for any face $F \in \mathcal{F}^\ell$ gives,

$$S_{\overline{\overline{F}}}^T Q_F^T \widetilde{A}^\ell Q_F S_{\overline{\overline{F}}} = \left[\begin{array}{cc|c} \widetilde{D}_{\overline{\overline{F}}} & & \\ & \widetilde{B}^\ell_{\widehat{F}\widehat{F}} & \left(\widetilde{A}^\ell_{R\widehat{F}}\right)^T \\ \hline & \widetilde{A}^\ell_{R\widehat{F}} & \widetilde{A}^\ell_{RR} \end{array}\right], \tag{21}$$

where $\widehat{F}$ is the skeleton DOFs of F, $\overline{\overline{F}}$ is the redundant DOFs of F, and R refers to the rest DOFs. Due to the elimination from previous levels, $|F|$ scales as $O(m2^\ell)$ and $\widetilde{A}^\ell_{RF}$ contains a nonzero submatrix of size $O(m2^\ell) \times O(m2^\ell)$. Therefore, the interpolative decomposition can be formed efficiently. Readers are referred to Sect. 2.3 for the explicit forms of each matrix in (21).

Looping over all faces at level ℓ, we obtain

$$\begin{aligned} A^{\ell+1} &\approx \left(\prod_{F\in\mathcal{F}^\ell} S_{\overline{\overline{F}}} Q_F\right)^T \widetilde{A}^\ell \left(\prod_{F\in\mathcal{F}^\ell} S_{\overline{\overline{F}}} Q_F\right) \\ &= \left(\prod_{F\in\mathcal{F}^\ell} S_{\overline{\overline{F}}} Q_F\right)^T \left(\prod_{I\in\mathcal{I}^\ell} S_I\right)^T A^\ell \left(\prod_{I\in\mathcal{I}^\ell} S_I\right) \left(\prod_{F\in\mathcal{F}^\ell} S_{\overline{\overline{F}}} Q_F\right) \\ &= \left(W^\ell\right)^T A^\ell W^\ell, \end{aligned} \tag{22}$$

where $W^\ell = \left(\prod_{I\in\mathcal{I}^\ell} S_I\right)\left(\prod_{F\in\mathcal{F}^\ell} S_{\overline{\overline{F}}} Q_F\right)$. The active DOFs for the next level are now defined as,

$$\Sigma^{\ell+1} = \widetilde{\Sigma}^\ell \setminus \bigcup_{F\in\mathcal{F}^\ell} \overline{\overline{F}} = \Sigma^\ell \setminus \left(\left(\bigcup_{I\in\mathcal{I}^\ell} I\right) \bigcup \left(\bigcup_{F\in\mathcal{F}^\ell} \overline{\overline{F}}\right)\right). \tag{23}$$

- *Level L* Finally, A^L and Σ^L are the matrix and active DOFs at level L. Up to a permutation, A^L can be factorized as

$$A^L = \begin{bmatrix} A^L_{\Sigma^L\Sigma^L} & \\ & D_R \end{bmatrix} = \begin{bmatrix} L_{\Sigma^L} & \\ & I \end{bmatrix} \begin{bmatrix} D_{\Sigma^L} & \\ & D_R \end{bmatrix} \begin{bmatrix} L^T_{\Sigma^L} & \\ & I \end{bmatrix} := \left(W^L\right)^{-T} D \left(W^L\right)^{-1}. \tag{24}$$

Combining all these factorization results

$$A \approx \left(W^0\right)^{-T} \cdots \left(W^{L-1}\right)^{-T} \left(W^L\right)^{-T} D \left(W^L\right)^{-1} \left(W^{L-1}\right)^{-1} \cdots \left(W^0\right)^{-1} \equiv F \tag{25}$$

and

$$A^{-1} \approx W^0 \cdots W^{L-1} W^L D^{-1} \left(W^L\right)^T \left(W^{L-1}\right)^T \cdots \left(W^0\right)^T = F^{-1}. \tag{26}$$

A^{-1} is factorized into a multiplicative sequence of matrices W^ℓ and each W^ℓ corresponding to level ℓ is again a multiplicative sequence of sparse matrices, S_I, $S_{\overline{\overline{F}}}$ and Q_F. Due to the fact that any S_I, $S_{\overline{\overline{F}}}$ or Q_F contains a small non-trivial (i.e., neither

identity nor empty) matrix of size $O(\frac{N^{1/3}}{2^{L-\ell}}) \times O(\frac{N^{1/3}}{2^{L-\ell}})$, the overall complexity for strong and applying W^ℓ is $O(N/2^\ell)$. Hence, the application of the inverse of A is of $O(N)$ computation and memory complexity.

3 Distributed-memory HIF

This section describes the algorithm for the distributed-memory HIF.

3.1 Process tree

For simplicity, assume that there are 8^L processes. We introduce a *process tree* that has $L+1$ levels and resembles the hierarchical structure of the computation domain. Each node of this process tree is called a *process group*. First at the leaf level, there are 8^L leaf process groups denoted as $\{p^0_{\mathbf{j}}\}_{\mathbf{j}}$. Here $\mathbf{j} = (j_1, j_2, j_3)$, $0 \le j_1, j_2, j_3 < 2^L$, and the superscript 0 refers to the leaf level (level 0). Each group at this level only contains a single process. Each node at level 1 of the process tree is constructed by merging 8 leaf processes. More precisely, we denote the process group at level 1 as $p^1_{\mathbf{j}}$ for $\mathbf{j} = (j_1, j_2, j_3)$, $0 \le j_1, j_2, j_3 < 2^{L-1}$, and $p^1_{\mathbf{j}} = \bigcup_{\lfloor \mathbf{j}_c/2 \rfloor = \mathbf{j}} p^0_{\mathbf{j}_c}$. Similarly, we recursively define the node at level ℓ as $p^\ell_{\mathbf{j}} = \bigcup_{\lfloor \mathbf{j}_c/2 \rfloor = \mathbf{j}} p^{\ell-1}_{\mathbf{j}_c}$. Finally, the process group $p^L_{\mathbf{0}}$ at the root includes all processes. Figure 4 illustrates the process tree. Each cube in the process tree is a process group.

3.2 Distributed-memory method

Same as in Sect. 2.4, we define the $n \times n \times n$ grid on $\Omega = [0,1)^3$ for $n = m2^L$, where $m = O(1)$ is a small positive integer and $L = O(\log N)$ is the level number of the root level. Discretizing (1) with seven-point stencil on the grid provides the linear system $Au = f$, where A is a sparse $N \times N$ SPD matrix, $u \in \mathbb{R}^N$ is the unknown function at grid points, and $f \in \mathbb{R}^N$ is the given function at grid points.

Given the process tree (Sect. 3.1) with 8^L processes and the sequential HIF structure (Sect. 2.4), the construction of the distributed-memory hierarchical interpolative factorization (DHIF) consists of the following steps.

- *Preliminary* Construct the process tree with 8^L processes. Each process group $p^0_{\mathbf{j}}$ owns the data corresponding to cell $C^0_{\mathbf{j}}$ and the set of active DOFs in $C^0_{\mathbf{j}}$ is denoted as $\Sigma^0_{\mathbf{j}}$, where $\mathbf{j} = (j_1, j_2, j_3)$ and $0 \le j_1, j_2, j_3 < 2^L$. Set $A^0 = A$ and let the process group $p^0_{\mathbf{j}}$ own $A^0(:, \Sigma^0_{\mathbf{j}})$, which is a sparse tall-skinny matrix with $O(N/P)$ nonzero entries.
- *Level ℓ for $\ell = 0, \ldots, L-1$.*

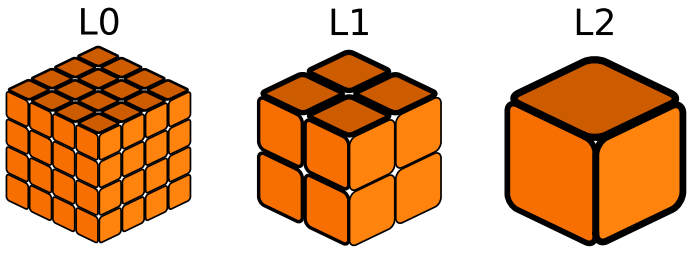

Fig. 4 Process tree: 64 processes are organized in the process tree

– *Preliminary* Let A^ℓ denote the matrix before any elimination at level ℓ. $\Sigma_{\mathbf{j}}^\ell$ denotes the active DOFs owned by the process group $p_{\mathbf{j}}^\ell$ for $\mathbf{j} = (j_1, j_2, j_3)$, $0 \le j_1, j_2, j_3 < 2^{L-\ell}$, and the nonzero submatrix of $A^\ell(:, \Sigma_{\mathbf{j}}^\ell)$ is distributed among the process group $p_{\mathbf{j}}^\ell$ using the two-dimensional block-cyclic distribution.
– *Sparse elimination* The process group $p_{\mathbf{j}}^\ell$ owns $A^\ell(:, \Sigma_{\mathbf{j}}^\ell)$, which is sufficient for performing sparse elimination for $I_{\mathbf{j}}^\ell$. To simplify the notation, we define $I = I_{\mathbf{j}}^\ell$ as the active interior DOFs of cell $C_{\mathbf{j}}^\ell$, $F = \mathcal{F}_{\mathbf{j}}^\ell$ as the surrounding faces, and $R = R_{\mathbf{j}}^\ell$ as the rest active DOFs. Sparse elimination at level ℓ within the process group $p_{\mathbf{j}}^\ell$ performs essentially

$$S_I^T A^\ell S_I = \begin{bmatrix} D_I & & \\ & B_{FF}^\ell & \left(A_{RF}^\ell\right)^T \\ & A_{RF}^\ell & A_{RR}^\ell \end{bmatrix}, \tag{27}$$

where $B_{FF}^\ell = A_{FF}^\ell - A_{FI}^\ell \left(A_{II}^\ell\right)^{-1} \left(A_{FI}^\ell\right)^T$,

$$S_I = \begin{bmatrix} \left(L_I^\ell\right)^{-T} & -\left(A_{II}^\ell\right)^{-1} \left(A_{FI}^\ell\right)^T & \\ & I & \\ & & I \end{bmatrix} \tag{28}$$

with $L_I^\ell D_I \left(L_I^\ell\right)^T = A_{II}^\ell$. Since $A^\ell(:, \Sigma_{\mathbf{j}}^\ell)$ is owned locally by $p_{\mathbf{j}}^\ell$, both A_{FI}^ℓ and A_{II}^ℓ are local matrices. All non-trivial (i.e., neither identity nor empty) submatrices in S_I are formed locally and stored locally for application. On the other hand, updating on A_{FF}^ℓ requires some communication in the next step.
– *Communication after sparse elimination* After all sparse eliminations are performed, some communication is required to update A_{FF}^ℓ for each cell $C_{\mathbf{j}}^\ell$. For the problem (1) with the periodic boundary conditions, each face at level ℓ is the surrounding face of exactly two cells. The owning process groups of these two cells need to communicate with each other to apply the additive updates, a submatrix of $-A_{FI}^\ell \left(A_{II}^\ell\right)^{-1} \left(A_{FI}^\ell\right)^T$. Once all communications are finished, the parallel sparse elimination does the rest of the computation, which can be conceptually denoted as

$$\begin{aligned} \widetilde{A}^\ell &= \left(\prod_{I \in \mathcal{I}^\ell} S_I \right)^T A^\ell \left(\prod_{I \in \mathcal{I}^\ell} S_I \right), \\ \widetilde{\Sigma}_{\mathbf{j}}^\ell &= \Sigma_{\mathbf{j}}^\ell \setminus \bigcup_{I \in \mathcal{I}^\ell} I, \end{aligned} \tag{29}$$

for $\mathbf{j} = (j_1, j_2, j_3)$, $0 \le j_1, j_2, j_3 < 2^{L-\ell}$.
– *Skeletonization* For each face F owned by $p_{\mathbf{j}}^\ell$, the corresponding matrices $\widetilde{A}^\ell(:, F)$ are stored locally. Similar to the parallel sparse elimination part, most operations are local at the process group $p_{\mathbf{j}}^\ell$ and can be carried out using the dense parallel linear algebra efficiently. By forming a parallel interpolative decomposition (ID) for $\widetilde{A}_{RF}^\ell = \left[\widetilde{A}_{R\widehat{F}}^\ell T_F^\ell \; \widetilde{A}_{R\widehat{F}}^\ell\right]$, the parallel skeletonization can be, conceptually, written as

$$S_{\overline{\overline{F}}} Q_F \widetilde{A}^{\ell} (Q_F)^T \left(S_{\overline{\overline{F}}}\right)^T \approx \left[\begin{array}{cc|c} D_{\overline{\overline{F}}} & & \\ & \widetilde{B}^{\ell}_{\widehat{F}\widehat{F}} & \widetilde{A}^{\ell}_{R\widehat{F}} \\ \hline & \widetilde{A}^{\ell}_{R\widehat{F}} & \widetilde{A}^{\ell}_{RR} \end{array}\right], \tag{30}$$

where the definitions of Q_F and $S_{\overline{\overline{F}}}$ are given in the discussion of skeletonization. Since $\widetilde{A}^{\ell}_{\overline{\overline{F}}\,\overline{\overline{F}}}$, $\widetilde{A}^{\ell}_{\widehat{F}\overline{\overline{F}}}$, $\widetilde{A}^{\ell}_{\widehat{F}\widehat{F}}$ and T^{ℓ}_F are all owned by $p^{\ell}_{\mathbf{j}}$, it requires only local operations to form

$$\begin{aligned} \widetilde{B}^{\ell}_{\overline{\overline{F}}\,\overline{\overline{F}}} &= \widetilde{A}^{\ell}_{\overline{\overline{F}}\,\overline{\overline{F}}} - \left(T^{\ell}_F\right)^T \widetilde{A}^{\ell}_{\widehat{F}\overline{\overline{F}}} - \left(\widetilde{A}^{\ell}_{\widehat{F}\overline{\overline{F}}}\right)^T T^{\ell}_F + \left(T^{\ell}_F\right)^T \widetilde{A}^{\ell}_{\widehat{F}\widehat{F}} T^{\ell}_F, \\ \widetilde{B}^{\ell}_{\widehat{F}\overline{\overline{F}}} &= \widetilde{A}^{\ell}_{\widehat{F}\overline{\overline{F}}} - \widetilde{A}^{\ell}_{\widehat{F}\widehat{F}} T^{\ell}_F, \\ \widetilde{B}^{\ell}_{\widehat{F}\widehat{F}} &= \widetilde{A}^{\ell}_{\widehat{F}\widehat{F}} - \widetilde{B}^{\ell}_{\widehat{F}\overline{\overline{F}}} \left(\widetilde{B}^{\ell}_{\overline{\overline{F}}\,\overline{\overline{F}}}\right)^{-1} \left(\widetilde{B}^{\ell}_{\widehat{F}\overline{\overline{F}}}\right)^T. \end{aligned} \tag{31}$$

Similarly, $L_{\overline{\overline{F}}}$, which is the LDL^T factor of $\widetilde{B}_{\overline{\overline{F}}\,\overline{\overline{F}}}$, is also formed within the process group $p^{\ell}_{\mathbf{j}}$. Moreover, since non-trivial blocks in Q_F and $S_{\overline{\overline{F}}}$ are both local, this implies that the applications of Q_F and $S_{\overline{\overline{F}}}$ are local operations. As a result, the parallel skeletonization factorizes A^{ℓ} conceptually as

$$\begin{aligned} A^{\ell+1} &\approx \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\overline{\overline{F}}} Q_F\right)^T \widetilde{A}^{\ell} \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\overline{\overline{F}}} Q_F\right) \\ &= \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\overline{\overline{F}}} Q_F\right)^T \left(\prod_{I\in\mathcal{I}^{\ell}} S_I\right)^T A^{\ell} \left(\prod_{I\in\mathcal{I}^{\ell}} S_I\right) \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\overline{\overline{F}}} Q_F\right) \end{aligned} \tag{32}$$

and we can define

$$\begin{aligned} W^{\ell} &= \left(\prod_{I\in\mathcal{I}^{\ell}} S_I\right) \left(\prod_{F\in\mathcal{F}^{\ell}} S_{\overline{\overline{F}}} Q_F\right), \\ \Sigma^{\ell+1/2}_{\mathbf{j}} &= \widetilde{\Sigma}^{\ell}_{\mathbf{j}} \setminus \bigcup_{F\in\mathcal{F}^{\ell}} \overline{\overline{F}} \\ &= \Sigma^{\ell}_{\mathbf{j}} \setminus \left(\left(\bigcup_{F\in\mathcal{F}^{\ell}} \overline{\overline{F}}\right) \bigcup \left(\bigcup_{I\in\mathcal{I}^{\ell}} I\right)\right). \end{aligned} \tag{33}$$

We would like to emphasize that the factors W^{ℓ} are evenly distributed among the process groups at level ℓ and that all non-trivial blocks are stored locally.

– *Merging and redistribution* Toward the end of the factorization at level ℓ, we need to merge the process groups and redistribute the data associated with the active DOFs in order to prepare for the work at level $\ell+1$. For each process group at level $\ell+1$, $p^{\ell+1}_{\mathbf{j}}$, for $\mathbf{j} = (j_1, j_2, j_3)$, $0 \le j_1, j_2, j_3 < 2^{L-\ell-1}$, we first form its active DOF set $\Sigma^{\ell+1}_{\mathbf{j}}$ by merging $\Sigma^{\ell+1/2}_{\mathbf{j}_c}$ from all its children $p^{\ell}_{\mathbf{j}_c}$, where $\lfloor \mathbf{j}_c/2 \rfloor = \mathbf{j}$. In addition, $A^{\ell+1}(:, s^{\ell+1}_{\mathbf{j}})$ is separately owned by $\{p^{\ell}_{\mathbf{j}_c}\}_{\lfloor \mathbf{j}_c/2 \rfloor = \mathbf{j}}$. A redistribution among $p^{\ell+1}_{\mathbf{j}}$ is needed in order to reduce the communication cost for future parallel dense linear algebra. Although this redistribution requires a global communication among $p^{\ell+1}_{\mathbf{j}}$, the complexities for message and bandwidth are bounded by the cost for parallel dense linear algebra. Actually, as we shall see in the numerical results, its cost is far lower than that of the parallel dense linear algebra.

- *Level L factorization* The parallel factorization at level L is quite similar to the sequential one. After factorizations from all previous levels, $A^L(\Sigma_{\mathbf{0}}^L, \Sigma_{\mathbf{0}}^L)$ is distributed among $p_{\mathbf{0}}^L$. A parallel LDL^T factorization of $A^L_{\Sigma_{\mathbf{0}}^L\Sigma_{\mathbf{0}}^L} = A^L(\Sigma_{\mathbf{0}}^L, \Sigma_{\mathbf{0}}^L)$ among the processes in $p_{\mathbf{0}}^L$ results

$$A^L = \begin{bmatrix} A^L_{\Sigma_{\mathbf{0}}^L\Sigma_{\mathbf{0}}^L} & \\ & D_R \end{bmatrix} = \begin{bmatrix} L^L_{\Sigma_{\mathbf{0}}^L} & \\ & I \end{bmatrix} \begin{bmatrix} D^L_{\Sigma_{\mathbf{0}}^L} & \\ & D_R \end{bmatrix} \begin{bmatrix} \left(L^L_{\Sigma_{\mathbf{0}}^L}\right)^T & \\ & I \end{bmatrix} = \left(W^L\right)^{-T} D \left(W^L\right)^{-1}. \tag{34}$$

  Consequently, we form the DHIF for A and A^{-1} as

$$A \approx \left(W^0\right)^{-T} \cdots \left(W^{L-1}\right)^{-T} \left(W^L\right)^{-T} D \left(W^L\right)^{-1} \left(W^{L-1}\right)^{-1} \cdots \left(W^0\right)^{-1} \equiv F \tag{35}$$

  and

$$A^{-1} \approx W^0 \cdots W^{L-1} W^L D^{-1} \left(W^L\right)^T \left(W^{L-1}\right)^T \cdots \left(W^0\right)^T = F^{-1}. \tag{36}$$

  The factors, W^ℓ are evenly distributed among all processes and the application of F^{-1} is basically a sequence of parallel dense matrix–vector multiplications.

In Fig. 5, we illustrate an example of DHIF for problem of size $24 \times 24 \times 24$ with $m = 6$ and $L = 2$. The computation is distributed on a process tree with $4^3 = 64$ processes. Particularly, Fig. 5 highlights the DOFs owned by process groups involving $p^0_{(0,1,0)}$, i.e., $p^0_{(0,1,0)}$, $p^1_{(0,0,0)}$, and $p^2_{(0,0,0)}$. Yellow points denote interior active DOFs, blue and brown points denote face active DOFs, and black points denote edge active DOFs. Meanwhile, we also have unfaded and faded groups of points. Unfaded points are owned by the process groups involving $p^0_{(0,1,0)}$. In other words, $p^0_{(0,1,0)}$ is the owner for part of the unfaded points. The faded points are owned by other process groups. In the second row and the fourth row, we also see faded brown points, which indicates the required communication to process $p^0_{(0,1,0)}$. Here Fig. 5 works through two levels of the elimination processes of the DHIF step by step.

3.3 Complexity analysis

3.3.1 Memory complexity

There are two places in the distributed algorithm that require heavy memory usage. The first one is to store the original matrix A and its updated version A^ℓ for each level ℓ. As we mentioned above in the parallel algorithm, A^ℓ contains at most $O(N)$ nonzeros and they are evenly distributed on P processes as follows. At level ℓ, there are $8^{L-\ell}$ cells, empirically each of which contains $O(\frac{N^{1/3}}{2^{L-\ell}})$ active DOFs. Meanwhile, each cell is evenly owned by a process group with 8^ℓ processes. Hence, $O((\frac{N^{1/3}}{2^{L-\ell}})^2)$ nonzero entries of $A^\ell(:, s^\ell_{\mathbf{j}})$ are evenly distributed on process group $p^\ell_{\mathbf{j}}$ with 8^ℓ processes. Overall, there are $O(8^{L-\ell} \cdot \frac{N^{2/3}}{4^{L-\ell}}) = O(N \cdot 2^{-\ell})$ nonzero entries in A^ℓ evenly distributed on $8^{L-\ell} \cdot 8^\ell = P$ processes, and each process owns $O(\frac{N}{P} \cdot 2^{-\ell})$ data for A^ℓ. Moreover, the factorization at level ℓ does not rely on the matrix $A^{\ell'}$ for $\ell' < \ell - 1$. Therefore, the memory cost for storing A^ℓs is $O(\frac{N}{P})$ for each process.

The second place is to store the factors W^ℓ. It is not difficult to see that the memory cost for each W^ℓ is the same as A^ℓ. Only non-trivial blocks in S_I, Q_F, and $S_{\overline{F}}$ require storage. Since each of these non-trivial blocks is of size $O(\frac{N^{1/3}}{2^{L-\ell}}) \times O(\frac{N^{1/3}}{2^{L-\ell}})$ and evenly distributed

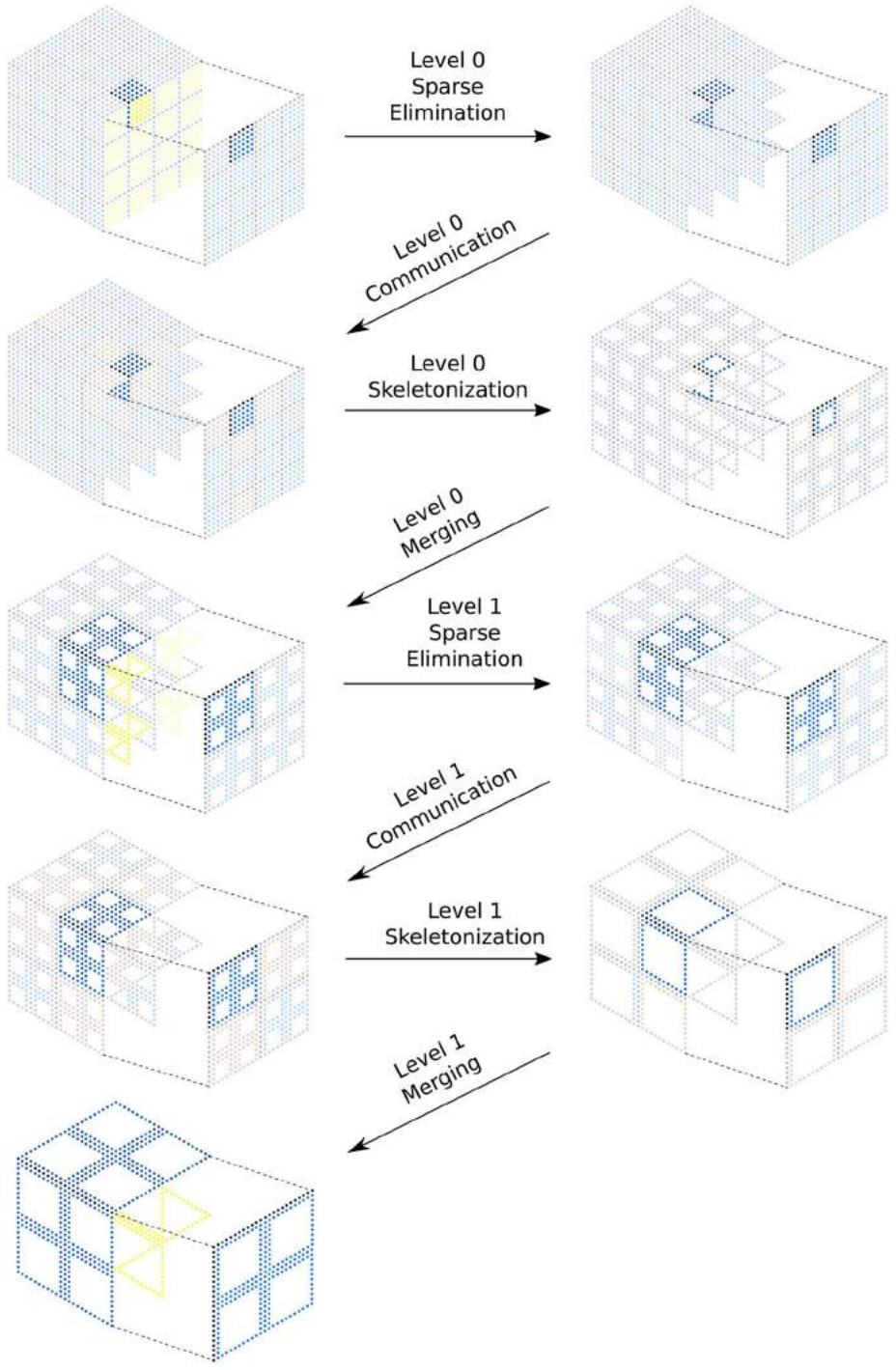

Fig. 5 Distributed-memory hierarchical interpolative factorization

on 8^ℓ processes, the overall memory requirement for each W^ℓ on a process is $O(\frac{N}{P} \cdot 2^{-\ell})$. Therefore, $O(\frac{N}{P})$ memory is required on each process to store all W^ℓs.

3.3.2 Computation complexity

The majority of the computation work goes to the construction of S_I, Q_F, and $S_{\overline{F}}$. As stated in the previous section, at level ℓ, each non-trivial dense matrix in these factors

is of size $O(\frac{N^{1/3}}{2^{L-\ell}}) \times O(\frac{N^{1/3}}{2^{L-\ell}})$. The construction adopts the matrix–matrix multiplication, the interpolative decomposition (pivoting QR), the LDL^T factorization, and the triangular matrix inversion. Each of these operation is of cubic computation complexities and the corresponding parallel computation cost over 8^ℓ processes is $O(\frac{N}{P})$. Since there are only a constant number of these operations per process at a single level, the total computational complexity across all $O(\log N)$ levels is $O(\frac{N \log N}{P})$.

The application computational complexity is simply the complexity of applying each nonzero entries in W^ℓs once, hence, the overall computational complexity is the same as the memory complexity $O(\frac{N}{P})$.

3.3.3 Communication complexity

The communication complexity is composed of three parts: the communication in the parallel dense linear algebra, the communication after sparse elimination, and the merging and redistribution step within DHIF. It is clear to see that the communication cost for the second part is bounded by either of the rest. Hence, we will simply derive the communication cost for the first and third parts. Here, we adopt the simplified communication model, $T_{\text{comm}} = \alpha + \beta$, where α is the latency and β is the inverse bandwidth.

At level ℓ, the parallel dense linear algebra involves the matrix–matrix multiplication, the ID, the LDL^T factorization, and the triangular matrix inversion for matrices of size $O(\frac{N^{1/3}}{2^{L-\ell}}) \times O(\frac{N^{1/3}}{2^{L-\ell}})$. All these basic operations are carried out on a process group of size 8^ℓ. Following the discussion in [5], the communication cost for these operations is bounded by $O(\ell^3\sqrt{8^\ell})\alpha + O(\frac{N^{2/3}}{4^{L-\ell}8^\ell}\ell)\beta$. Summing over all levels, one can control the communication cost of the parallel dense linear algebra part by

$$O\left(\sqrt{P}\log^3 P\right)\alpha + O\left(\frac{N^{2/3}}{P^{2/3}}\right)\beta. \tag{37}$$

On the other hand, the merging and redistribution step at level ℓ involves $8^{\ell+1}$ processes and redistributes matrices of size $O(\frac{N^{1/3}}{2^{L-\ell}} \cdot 8) \times O(\frac{N^{1/3}}{2^{L-\ell}} \cdot 8)$. The current implementation adopts the MPI routine MPI_AllToAll to handle the redistribution on a 2D process mesh. Further, we assume the all-to-all communication sends and receives long messages. The standard upper bound for the cost of this routine is $O(\sqrt{8^{\ell+1}})\alpha + O(\frac{N^{2/3}}{4^{L-\ell}\sqrt{8^{\ell+1}}} \cdot 64)\beta$ [37]. Therefore, the overall cost is

$$O\left(\sqrt{P}\right)\alpha + O\left(\frac{N^{2/3}}{\sqrt{P}}\right)\beta. \tag{38}$$

The complexity of the latency part is not scalable. However, empirically, the cost for this communication is relatively small in the actual running time.

4 Numerical results

Here we present a couple of numerical examples to demonstrate the parallel efficiency of the distributed-memory HIF. The algorithm is implemented in C++11 and all inter-process communication is expressed via the message passing interface (MPI). The distributed-memory dense linear algebra computation is done through the Elemental library [31]. All numerical examples are performed on Edison at the National Energy Research Scientific Computing center (NERSC). The numbers of processes used are always powers of two, ranging from 1 to 8192. The memory allocated for each process is limited to 2 GB.

All numerical results are measured in two ways: the strong scaling and weak scaling. The strong scaling measurement fixes the problem size and increases the number of processes. For a fixed problem size, let T_P^S be the running time of P processes. The strong scaling efficiency is defined as,

$$E^S = \frac{T_1^S}{P \cdot T_P^S}. \tag{39}$$

In the case that T_1^S is not available, e.g., the fixed problem cannot fit into the single process memory, we adopt the first available running time, T_m^S, associating with the smallest number of processes, m, as a reference. And the modified strong scaling efficiency is,

$$E^S = \frac{m \cdot T_m^S}{P \cdot T_P^S}. \tag{40}$$

The weak scaling measurement fixes the ratio between the problem size and the number of processes. For a fixed ratio, we define the weak scaling efficiency as,

$$E^W = \frac{T_m^W}{T_P^W}, \tag{41}$$

where T_m^W is the first available running time with m processes and T_P^W is the running time of P processes.

The notations used in the following tables and figures are listed in Table 2. For simplicity, all examples are defined over $\Omega = [0, 1)^3$ with periodic boundary condition, discretized on a uniform grid, $n \times n \times n$, with n being the number of points in each dimension and $N = n^3$. The PDEs defined in (1) are discretized using the second-order central difference method with seven-point stencil, which is the same as (16). Octrees are adopted to partition the computation domain with the block size at leaf level bounded by 64.

Example 1 We first consider the problem in (1) with $a(x) \equiv 1$ and $b(x) \equiv 0.1$. The relative precision of the ID is set to be $\epsilon = 10^{-3}$.

As shown in Table 3, given the tolerance $\epsilon = 10^{-3}$ the relative error remains well below this for all N and P. The number of skeleton points on the root level, $|\Sigma_L|$, grows linearly as the one-dimensional problem size increases. The empirical linear scaling of the root skeleton size strongly supports the quasi-linear scaling for the factorization, linear scaling

Table 2 Notations for the numerical results

Notation	Explanation
ϵ	Relative precision of the ID
N	Total number of DOFs in the problem
e_s	Relative error for solving, $\left\|(I - F^{-1}A)x\right\| / \|x\|$, where x is a Gaussian random vector
$\|\Sigma_L\|$	Number of remaining active DOFs at the root level
m_f	Maximum memory required to perform the factorization in GB across all processes
t_f	Time for constructing the factorization in seconds
E_f^S	Strong scaling efficiency for factorization time
t_s	Time for applying F^{-1} to a vector in seconds
E_s^S	Strong scaling efficiency for application time
n_{iter}	Number of iterations to solve $Au = f$ with GMRES with F^{-1} being a preconditioner to a tolerance of 10^{-12}

Table 3 Example 1: numerical results

N	P	e_s	$\|s_L\|$	m_f	t_f	$E_f^S(\%)$	t_s	$E_s^S(\%)$	n_{iter}
32^3	1	4.84e−04	3440	1.92e−01	4.85e+00	100	1.36e−01	100	6
	2	5.26e−04	3440	9.60e−02	2.60e+00	93	6.65e−02	103	6
	4	3.78e−04	3440	4.80e−02	1.45e+00	84	3.47e−02	98	6
	8	4.93e−04	3440	2.40e−02	8.38e−01	72	1.99e−02	85	6
	16	3.97e−04	3440	1.20e−02	5.83e−01	52	1.31e−02	65	6
	32	7.33e−04	3440	6.03e−03	4.35e−01	35	1.47e−02	29	6
64^3	2	5.92e−04	7760	9.07e−01	3.87e+01	100	6.08e−01	100	6
	4	5.98e−04	7760	4.54e−01	2.36e+01	82	2.99e−01	102	6
	8	5.59e−04	7760	2.27e−01	1.48e+01	65	1.61e−01	94	6
	16	6.30e−04	7760	1.13e−01	1.03e+01	47	9.52e−02	80	6
	32	5.89e−04	7760	5.68e−02	5.34e+00	45	6.88e−02	55	6
	64	5.45e−04	7760	2.84e−02	2.67e+00	45	4.10e−02	46	6
	128	5.43e−04	7760	1.42e−02	1.52e+00	40	3.43e−02	28	6
	256	6.29e−04	7760	7.14e−03	1.27e+00	24	2.69e−02	18	6
128^3	16	6.19e−04	16,208	9.77e−01	1.43e+02	100	8.24e−01	100	6
	32	5.98e−04	16,208	4.89e−01	7.40e+01	97	4.37e−01	94	6
	64	5.85e−04	16,208	2.44e−01	3.87e+01	92	2.26e−01	91	6
	128	6.23e−04	16,208	1.22e−01	2.11e+01	85	1.40e−01	74	6
	256	6.14e−04	16,208	6.12e−02	1.00e+01	89	9.76e−02	53	6
	512	5.96e−04	16,208	3.06e−02	5.80e+00	77	1.98e−01	13	6
	1024	5.86e−04	16,208	1.54e−02	3.46e+00	65	6.13e−02	21	6
256^3	128	6.18e−04	33,104	1.01e+00	2.24e+02	100	9.18e−01	100	6
	256	6.11e−04	33,104	5.07e−01	1.19e+02	94	4.88e−01	94	6
	512	6.06e−04	33,104	2.53e−01	6.33e+01	88	2.85e−01	81	6
	1024	6.25e−04	33,104	1.27e−01	3.19e+01	88	1.86e−01	62	6
	2048	6.18e−04	33,104	6.35e−02	2.44e+01	57	1.58e−01	36	6
	4096	6.16e−04	33,104	3.18e−02	1.27e+01	55	1.73e−01	17	6
	8192	6.14e−04	33,104	1.60e−02	1.16e+01	30	4.14e−01	3	6
512^3	1024	6.16e−04	66,896	1.03e+00	3.32e+02	100	1.08e+00	100	6
	2048	6.15e−04	66,896	5.16e−01	1.84e+02	90	6.53e−01	82	6
	4096	6.14e−04	66,896	2.58e−01	9.55e+01	87	4.90e−01	55	6
	8192	6.13e−04	66,896	1.29e−01	5.58e+01	74	4.58e−01	29	6
1024^3	8192	6.15e−04	134,480	1.04e+00	4.67e+02	100	1.48e+00	100	6

for the application, and linear scaling for memory cost. The column labeled with m_f in Table 3, or alternatively Fig. 6c, illustrates the perfect strong scaling for the memory cost. Since the bottleneck for most parallel algorithms is the memory cost, this point is especially important in practice. Perfect distribution of the memory usage allows us to solve very large problem on a massive number of processes, even through the communication penalty on massive parallel computing would be relatively large. The factorization time and application time show good scaling up to thousands of processes. Figure 6a, b presents the strong scaling plot for the running time of factorization and application, respectively. Together with Fig. 6d, which illustrates the timing for each part of the factorization, we conclude that the communication cost beside the parallel dense linear algebra (labeled with "El") remains small comparing to the cost of the parallel dense linear algebra. It is the parallel dense linear algebra part that stops the strong scaling. As it is also well known that parallel dense linear algebra achieves good weak scaling, so does our DHIF imple-

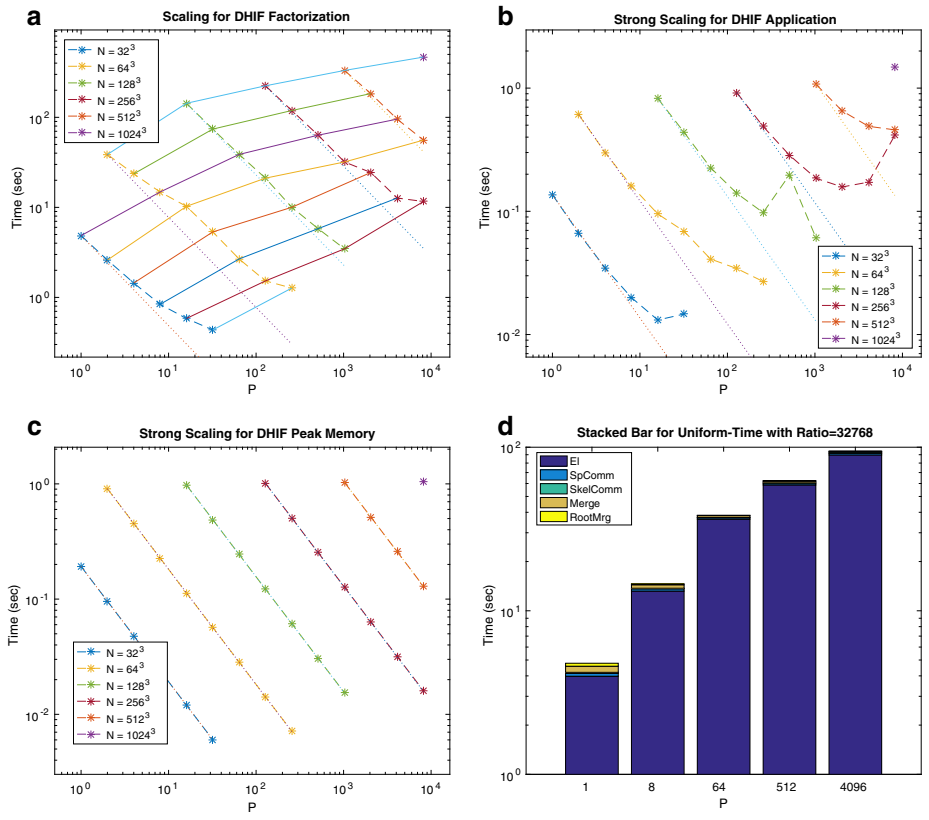

Fig. 6 Example 1. **a** is the scaling plot for the DHIF factorization time, the *solid lines* indicate the weak scaling results, the *dashed lines* are the strong scaling results, and the *dotted lines* are the reference lines for perfect strong scaling; the line style applies to all figures below; **b** is the strong scaling for the DHIF application time; **c** is the strong scaling for the DHIF peak memory usage; **d** shows a stacked *bar plot* for factorization time for fixed ratio between the problem size and the number of processes

mentation. Finally, the last column of Table 3 shows the number of iterations for solving $Au = f$ using the GMRES algorithm with a relative tolerance of 10^{-12} and with the DHIF as a preconditioner. As the numbers in the entire column are equal to 6, this shows that DHIF serves an excellent preconditioner with the iteration number almost independent of the problem size.

Example 2 The second example is a problem of (1) with high-contrast random field $a(x)$ and $b(x) \equiv 0.1$. The high-contrast random field $a(x)$ is defined as follows,

1. Generate uniform random value $a_{\mathbf{j}}$ between 0 and 1 for each discretization point;
2. Convolve the random value $a_{\mathbf{j}}$ with an isotropic three-dimensional Gaussian with standard deviation 1;
3. Quantize the field via

$$a_{\mathbf{j}} = \begin{cases} 0.1, & a_{\mathbf{j}} \leq 0.5 \\ 1000, & a_{\mathbf{j}} > 0.5 \end{cases}. \tag{42}$$

The given tolerance is set to be 10^{-5}.

Figure 7 shows a slice in a realization of the random field. The corresponding matrix A is clearly of high contrast. Solving such a problem is harder than Example 1 due to the raise of the condition number. The performance results of our algorithm are presented in Table 4. As we expect, the relative error for solving is lower than that in Table 3 and the number of iterations in GMRES is higher.

Table 4 and Fig. 8 demonstrate the efficiency of the DHIF for high-contrast random field. Almost all comments regarding the numerical results in Example 1 apply here. To focus on the difference between Example 1 and Example 2, the most noticeable difference is about the relative error, e_s. Though we give a higher relative precision $\epsilon = 10^{-5}$, the relative error for Example 2 is about 3×10^{-3}, which is about ten times larger than e_s in Example 1. The reason for the decrease of accuracy is most likely the increase of the condition number for Example 2. This also increases the number of iterations in GMRES. However, both e_s and n_{iter} remain roughly constant for varying problem sizes. This means that DHIF still serves as a robust and efficient solver and preconditioner for such problems. Another difference is the number of skeleton points on the root level, $|\Sigma_L|$. Due to the fact that the field $a(x)$ is random, and different rows in Table 4 actually adopt different realizations, the small fluctuation of $|\Sigma_L|$ for the same N and different P is expected. Overall $|\Sigma_L|$ still grows linearly as $n = N^{1/3}$ increases. This again supports the complexity analysis given above.

Example 3 The third example provides a concrete comparison between the proposed DHIF and multigrid method (hypre [13]). The problem behaves similar as example 2 without randomness, (1) with high-contrast field $a(x)$ and $b(x) \equiv 0.1$. The high-contrast field $a(x)$ is defined as follows,

$$a(\mathbf{x}) = \begin{cases} 1000, & \sum_{i=1}^{3} \lfloor \frac{x_i n}{7} \rfloor \equiv 0 \pmod 2, \\ 0.1, & \sum_{i=1}^{3} \lfloor \frac{x_i n}{7} \rfloor \equiv 1 \pmod 2, \end{cases} \tag{43}$$

where n is the number of grid points on each dimension.

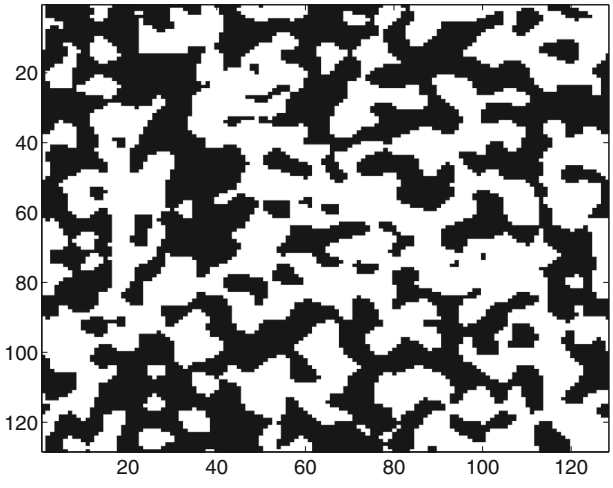

Fig. 7 A slice in a random field realization of size 128^3

Table 4 Example 2: numerical results

N	P	e_s	$\|s_L\|$	m_f	t_f	$E_f^S(\%)$	t_s	$E_s^S(\%)$	n_{iter}
32^3	1	3.02e−03	3865	2.00e−01	5.80e+00	100	1.34e−01	100	7
	2	3.39e−03	3632	9.31e−02	2.48e+00	117	6.69e−02	100	7
	4	2.69e−03	3934	5.13e−02	1.72e+00	84	3.75e−02	89	7
	8	3.18e−03	3660	2.37e−02	9.50e−01	76	2.20e−02	76	7
	16	3.13e−03	3693	1.24e−02	6.22e−01	58	1.32e−02	63	7
	32	3.00e−03	3744	6.42e−03	4.83e−01	38	1.49e−02	28	7
64^3	2	3.29e−03	8580	9.45e−01	4.33e+01	100	6.15e−01	100	7
	4	3.13e−03	8938	4.94e−01	2.91e+01	74	3.10e−01	99	7
	8	3.09e−03	9600	2.51e−01	1.98e+01	55	1.68e−01	91	7
	16	3.07e−03	8919	1.19e−01	1.27e+01	43	9.86e−02	78	7
	32	3.09e−03	9478	6.59e−02	6.99e+00	39	7.89e−02	49	7
	64	3.18e−03	9111	3.03e−02	3.17e+00	43	4.90e−02	39	7
	128	3.02e−03	9419	1.58e−02	2.15e+00	31	3.31e−02	29	7
	256	3.03e−03	9349	7.97e−03	1.60e+00	21	3.66e−02	13	7
128^3	16	3.16e−03	19,855	1.07e+00	2.11e+02	100	8.89e−01	100	7
	32	3.09e−03	20,487	5.58e−01	1.18e+02	90	4.86e−01	91	7
	64	3.06e−03	21,345	2.78e−01	6.43e+01	82	2.45e−01	91	7
	128	3.10e−03	20,344	1.37e−01	3.39e+01	78	1.34e−01	83	7
	256	3.07e−03	21,152	7.43e−02	1.76e+01	75	1.10e−01	51	7
	512	3.07e−03	20,779	3.51e−02	8.46e+00	78	8.80e−02	32	7
	1024	3.04e−03	21,361	1.76e−02	5.38e+00	61	6.31e−02	22	7
256^3	128	3.11e−03	42,420	1.14e+00	4.15e+02	100	1.04e+00	100	7
	256	3.12e−03	43,828	5.91e−01	2.12e+02	98	5.77e−01	90	8
	512	3.11e−03	44,126	2.90e−01	1.25e+02	83	3.86e−01	67	7
	1024	3.08e−03	43,302	1.46e−01	6.31e+01	82	2.12e−01	61	7
	2048	3.09e−03	44,131	7.78e−02	3.43e+01	76	1.86e−01	35	7
	4096	3.10e−03	43,691	3.71e−02	1.96e+01	66	2.28e−01	14	7
	8192	3.10e−03	43,952	1.85e−02	2.05e+01	32	4.03e−01	4	7
512^3	1024	3.11e−03	88,070	1.16e+00	6.37e+02	100	1.22e+00	100	7
	2048	3.11e−03	88,577	6.11e−01	3.47e+02	92	6.84e−01	89	8
	4096	3.11e−03	88,757	3.03e−01	1.89e+02	84	5.31e−01	58	7
	8192	3.11e−03	85,877	1.50e−01	1.02e+02	78	6.20e−01	25	7
1024^3	8192	3.11e−03	177,323	1.18e+00	9.35e+02	100	1.95e+00	100	8

We adopt GMRES iterative method in both DHIF and hypre to solve the elliptic problem to a relative error 10^{-12}. The given tolerance in DHIF is set to be 10^{-4}. And SMG interface in hypre is used as preconditioner for the problem on regular grids. The numerical results for DHIF and hypre are given in Table 5.

As we can read from Table 5, there are a few advantages of DHIF over hypre in the given settings. First, DHIF is faster than hypre's SMG except for small problems with small numbers of processes. And the number of iterations grows as the problem size grows in hypre, while it remains almost the same in DHIF. In truly large problems, the advantages of DHIF are more pronounced. Second, the scalability of DHIF appears to be better than that of hypre's SMG. Finally, DHIF only requires powers of two numbers of processes, whereas hypre's SMG requires powers of eight for 3D problems.

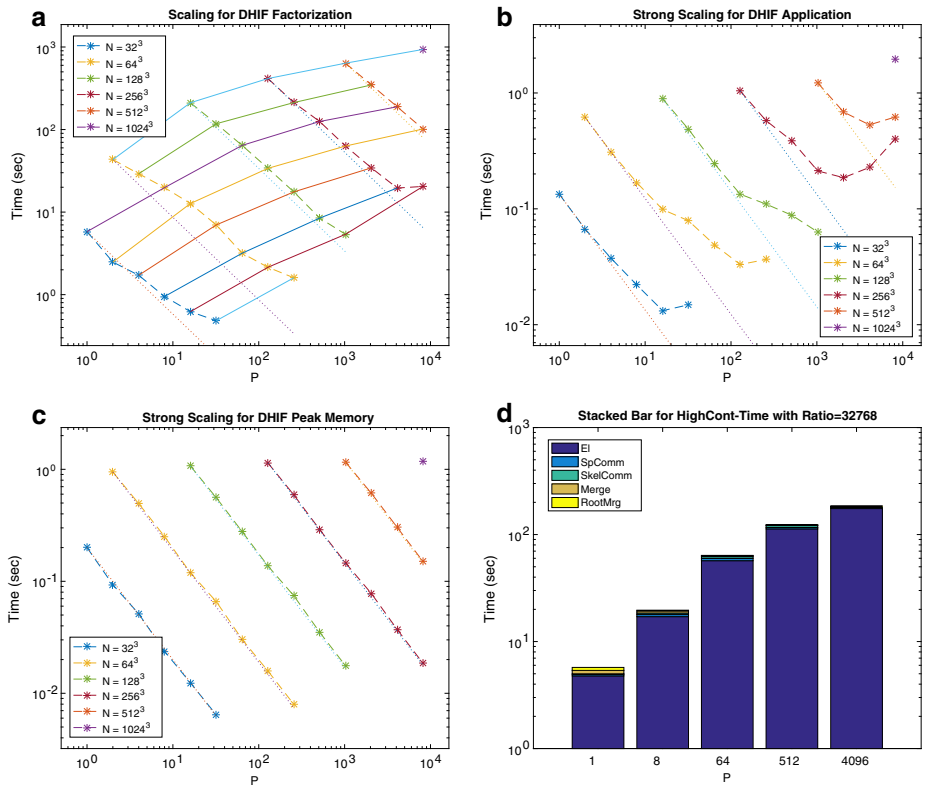

Fig. 8 Example 2. **a** provides a scaling plot for DHIF factorization time; **b** is the strong scaling for DHIF application time; **c** is the strong scaling for DHIF peak memory usage; **d** shows a stacked *bar plot* for factorization time for fixed ratio between problem size and number of processes

Table 5 Numerical results for DHIF and hypre

N	P	DHIF			hypre		
		$t_{\text{setup}}(s)$	$t_{\text{solve}}(s)$	n_{iter}	$t_{\text{setup}}(s)$	$t_{\text{solve}}(s)$	n_{iter}
64^3	8	15.27	18.10	21	0.29	9.67	67
	64	2.46	3.45	21	1.47	17.37	60
128^3	64	29.20	24.53	22	1.78	140.90	394
	512	3.93	4.41	22	2.11	113.57	455
256^3	512	59.66	26.33	21	4.11	258.22	492
	4096	11.58	6.78	21	8.97	191.15	375

t_{setup} is the setup time which is identical to t_f in previous examples for DHIF; t_{solve} is the total iterative solving time using GMRES; n_{iter} is the number of iterations in GMRES

5 Conclusion

In this paper, we introduced the distributed-memory hierarchical interpolative factorization (DHIF) for solving discretized elliptic partial differential equations in 3D. The computational and memory complexity for DHIF is

$$O\left(\frac{N\log N}{P}\right) \quad \text{and} \quad O\left(\frac{N}{P}\right), \tag{44}$$

respectively, where N is the total number of DOFs and P is the number of processes. The communication cost is

$$O\left(\sqrt{P}\log^3 P\right)\alpha + O\left(\frac{N^{2/3}}{\sqrt{P}}\right)\beta, \tag{45}$$

where α is the latency and β is the inverse bandwidth. Not only the factorization is efficient, the application can also be done in $O(\frac{N}{P})$ operations. Numerical examples in Sect. 4 illustrate the efficiency and parallel scaling of the algorithm. The results show that DHIF can be used both as a direct solver and as an efficient preconditioner for iterative solvers.

We have described the algorithm using the periodic boundary condition in order to simplify the presentation. However, the implementation can be extended in a straightforward way to problems with other type of boundary conditions. The discretization adopted here is the standard Cartesian grid. For more general discretizations such as finite element methods on unstructured meshes, one can generalize the current implementation by combining with the idea proposed in [36].

Here we have only considered the parallelization of the HIF for differential equations. As shown in [22], the HIF is also applicable to solving integral equations with non-oscillatory kernels. Parallelization of this algorithm is also of practical importance.

Author details
[1]ICME, Stanford University, Stanford, CA, USA, [2]Department of Mathematics, Stanford University, Stanford, CA, USA.

Acknowledgements
Y. Li and L. Ying are partially supported by the National Science Foundation under award DMS-1521830 and the U.S. Department of Energy's Advanced Scientific Computing Research program under award DE-FC02-13ER26134/DE-SC0009409. The authors would like to thank K. Ho, V. Minden, A. Benson, and A. Damle for helpful discussions. We especially thank J. Poulson for the parallel dense linear algebra package *Elemental*. We acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin (URL: http://www.tacc.utexas.edu) for providing HPC resources that have contributed to the research results reported in the early versions of this paper. This research, in the current version, used resources of the National Energy Research Scientific Computing Center, a DOE Office of Science User Facility supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

Received: 1 July 2016 Accepted: 27 February 2017
Published online: 05 June 2017

References

1. Ambikasaran, S., Darve, E.: An $\mathcal{O}(N \log N)$ fast direct solver for partial hierarchically semi-separable matrices: with application to radial basis function interpolation. SIAM J. Sci. Comput. **57**(3), 477–501 (2013)
2. Amestoy, P., Ashcraft, C., Boiteau, O., Buttari, A., L'Excellent, J.-Y., Weisbecker, C.: Improving multifrontal methods by means of block low-rank representations. SIAM J. Sci. Comput. **37**(3), A1451–A1474 (2015)
3. Amestoy, P.R., Duff, I.S., L'Excellent, J.-Y.: Multifrontal parallel distributed symmetric and unsymmetric solvers. Comput. Methods Appl. Mech. Eng. **184**(24), 501–520 (2000)
4. Amestoy, P.R., Duff, I.S., L'Excellent, J.-Y., Koster, J.: A fully asynchronous multifrontal solver using distributed dynamic scheduling. SIAM J. Matrix Anal. Appl. **23**(1), 15–41 (2001)
5. Ballard, G., Demmel, J., Holtz, O., Schwartz, O.: Minimizing communication in numerical linear algebra. SIAM J. Matrix Anal. Appl. **32**(3), 866–901 (2011)
6. Bebendorf, M.: Efficient inversion of the Galerkin matrix of general second-order elliptic operators with nonsmooth coefficients. Math. Comput. **74**(251), 1179–1199 (2005)
7. Bebendorf, M., Hackbusch, W.: Existence of $\mathcal{H}$-matrix approximants to the inverse FE-matrix of elliptic operators with L^∞-coefficients. Numer. Math. **95**(1), 1–28 (2003)
8. Börm, S.: Approximation of solution operators of elliptic partial differential equations by $\mathcal{H}$- and $\mathcal{H}^2$-matrices. Numer. Math. **115**(2), 165–193 (2010)
9. Briggs, W.L., Henson, V.E., McCormick, S.F.: A Multigrid Tutorial, 2nd edn. Society for Industrial and Applied Mathematics (2000). doi:10.1137/1.9780898719505
10. Chandrasekaran, S., Dewilde, P., Gu, M., Pals, T., Sun, X., van der Veen, A.-J., White, D.: Some fast algorithms for sequentially semiseparable representations. SIAM J. Matrix Anal. Appl. **27**(2), 341–364 (2005)
11. Chandrasekaran, S., Dewilde, P., Gu, M., Somasunderam, N.: On the numerical rank of the off-diagonal blocks of Schur complements of discretized elliptic PDEs. SIAM J. Matrix Anal. Appl. **31**(5), 2261–2290 (2010)
12. Cheng, H., Gimbutas, Z., Martinsson, P.-G., Rokhlin, V.: On the compression of low rank matrices. SIAM J. Sci. Comput. **26**(4), 1389–1404 (2005)

13. Chow, E., Falgout, R.D., Hu, J.J., Tuminaro, R.S., and Yang, U.M.: A survey of parallelization techniques for multigrid solvers. Parallel Process. Sci. Comput., chapter 10, pp. 179–201. Society for Industrial and Applied Mathematics (2006)
14. Duff, I.S., Reid, J.K.: The multifrontal solution of indefinite sparse symmetric linear equations. ACM Trans. Math. Softw. **9**(3), 302–325 (1983)
15. Falgout, R.D., Jones, J.E.: Multigrid on massively parallel architectures. In: Dick, E., Riemslagh, K., Vierendeels, J. (eds.) Multigrid Methods VI. Lecture Notes in Computational Science and Engineering, vol. 14, pp. 101–107. Springer, Berlin (2000). doi:10.1007/978-3-642-58312-4_13
16. George, A.: Nested dissection of a regular finite element mesh. SIAM J. Numer. Anal. **10**(2), 345–363 (1973)
17. Gillman, A., Martinsson, P.-G.: A direct solver with $O(N)$ complexity for variable coefficient elliptic PDEs discretized via a high-order composite spectral collocation method. SIAM J. Sci. Comput. **36**(4), A2023–A2046 (2014)
18. Hackbusch, W.: A sparse matrix arithmetic based on $\mathcal{H}$-matrices. I. Introduction to $\mathcal{H}$-matrices. Computing **62**(2), 89–108 (1999)
19. Hackbusch, W., Börm, S.: Data-sparse approximation by adaptive $\mathcal{H}^2$-matrices. Computing **69**(1), 1–35 (2002)
20. Hackbusch, W., Khoromskij, B.N.: A sparse $\mathcal{H}$-matrix arithmetic. II. Application to multi-dimensional problems. Computing **64**(1), 21–47 (2000)
21. Hao, S., Martinsson, P.-G.: A direct solver for elliptic PDEs in three dimensions based on hierarchical merging of Poincaré-Steklov operators. J. Comput. Appl. Math. **308**, 419–434 (2016). doi:10.1016/j.cam.2016.05.013
22. Ho, K.L., Ying, L.: Hierarchical interpolative factorization for elliptic operators: differential equations. Commun. Pure Appl. Math. **69**(8), 1415–1451 (2016). doi:10.1002/cpa.21582
23. Ho, K.L., Ying, L.: Hierarchical interpolative factorization for elliptic operators: integral equations. Commun. Pure Appl. Math. **69**(7), 1314–1353 (2016)
24. Izadi, M.: Parallel $\mathcal{H}$-matrix arithmetic on distributed-memory systems. Comput. Vis. Sci. **15**(2), 87–97 (2012)
25. Kriemann, R.: $\mathcal{H}$-LU factorization on many-core systems. Comput. Vis. Sci. **16**(3), 105 (2013)
26. Liu, J.W.H.: The multifrontal method for sparse matrix solution: theory and practice. SIAM Rev. **34**(1), 82–109 (1992)
27. Liu, X., Xia, J., Hoop, M.V.D.E.: Parallel randomized and matrix-free direct solvers for large structured dense linear systems. SIAM J. Sci. Comput. **38**(5), 1–32 (2016)
28. Martinsson, P.-G.: A fast direct solver for a class of elliptic partial differential equations. SIAM J. Sci. Comput. **38**(3), 316–330 (2009)
29. Martinsson, P.G.: Blocked rank-revealing QR factorizations: how randomized sampling can be used to avoid single-vector pivoting. arXiv:1505.08115 (2015)
30. Poulson, J., Engquist, B., Li, S., Ying, L.: A parallel sweeping preconditioner for heterogeneous 3D Helmholtz equations. SIAM J. Sci. Comput. **35**(3), C194–C212 (2013)
31. Poulson, J., Marker, B., van de Geijn, R.A., Hammond, J.R., Romero, N.A.: Elemental: a new framework for distributed memory dense matrix computations. ACM Trans. Math. Softw. **39**(2), 13:1–13:24 (2013)
32. Pouransari, H., Coulier, P., Darve, E.: Fast hierarchical solvers for sparse matrices using low-rank approximation. arXiv:1510.07363 (2016)
33. Saad, Y.: Parallel iterative methods for sparse linear systems. Stud. Comput. Math. **8**, 423–440 (2001)
34. Saad, Y.: Iterative Methods for Sparse Linear Systems, 2nd edn. Society for Industrial and Applied Mathematics (2003). doi:10.1137/1.9780898718003
35. Schmitz, P.G., Ying, L.: A fast direct solver for elliptic problems on general meshes in 2D. J. Comput. Phys. **231**(4), 1314–1338 (2012)
36. Schmitz, P.G., Ying, L.: A fast nested dissection solver for Cartesian 3D elliptic problems using hierarchical matrices. J. Comput. Phys. **258**, 227–245 (2014)
37. Scott, D.S.: Efficient all-to-all communication patterns in hypercube and mesh topologies. In: Distributed Memory Computing Conference, pp. 398–403. IEEE (1991)
38. Wang, S., Li, X.S., Rouet, F.H., Xia, J., De Hoop, M.V.: A parallel geometric multifrontal solver using hierarchically semiseparable structure. ACM Trans. Math. Softw. **42**(3), 21:1–21:21 (2016)
39. Xia, J.: Efficient structured multifrontal factorization for general large sparse matrices. SIAM J. Sci. Comput. **35**(2), A832–A860 (2013)
40. Xia, J.: Randomized sparse direct solvers. SIAM J. Matrix Anal. Appl. **34**(1), 197–227 (2013)
41. Xia, J., Chandrasekaran, S., Gu, M., Li, X.S.: Superfast multifrontal method for large structured linear systems of equations. SIAM J. Matrix Anal. Appl. **31**(3), 1382–1411 (2009)
42. Xia, J., Chandrasekaran, S., Gu, M., Li, X.S.: Fast algorithms for hierarchically semiseparable matrices. Numer. Linear Algebr. Appl. **17**(6), 953–976 (2010)
43. Xin, Z., Xia, J., De Hoop, M.V., Cauley, S., Balakrishnan, V.: A distributed-memory randomized structured multifrontal method for sparse direct solutions. Purdue GMIG Rep. **14**(17), 1–25 (2014)