# EFFICIENT SCALABLE ALGORITHMS FOR HIERARCHICALLY SEMISEPARABLE MATRICES

SHEN WANG*, XIAOYE S. LI†, JIANLIN XIA‡, YINGCHONG SITU§, AND MAARTEN V. DE HOOP¶

**Abstract.** Hierarchically semiseparable (HSS) matrix algorithms are emerging techniques in constructing the superfast direct solvers for both dense and sparse linear systems. Here, we develope a set of novel parallel algorithms for the key HSS operations that are used for solving large linear systems. These include the parallel rank-revealing QR factorization, the HSS constructions with hierarchical compression, the ULV HSS factorization, and the HSS solutions. The HSS tree based parallelism is fully exploited at the coarse level. The `BLACS` and `ScaLAPACK` libraries are used to facilitate the parallel dense kernel operations at the fine-grained level. We have appplied our new parallel HSS-embedded multifrontal solver to the anisotropic Helmholtz equations for seismic imaging, and were able to solve a linear system with 6.4 billion unknowns using 4096 processors, in about 20 minutes. The classical multifrontal solver simply failed due to high demand of memory. To our knowledge, this is the first successful demonstration of employing the HSS algorithms in solving the truly large-scale real-world problems. Our parallel strategies can be easily adapted to the parallelization of the other rank structured methods.

**Key words.** HSS matrix, parallel HSS algorithm, low-rank property, compression, HSS construction, direct solver

**AMS subject classifications.** 15A23, 65F05, 65F30, 65F50

## 1. Introduction.

In recent years, rank structured matrices have attracted much attention and have been widely used in the fast solutions of various partial differential equations, integral equations, and eigenvalue problems. Several useful rank structured matrix representations have been developed, such as $\mathcal{H}$-matrices [15, 18, 16], $\mathcal{H}^2$-matrices [4, 5, 17], quasiseparable matrices [1, 10], and semiseparable matrices [6, 24].

Here, we focus on a type of semiseparable structures, called *hierarchically semiseparable* (HSS) forms, in the context of fast direct solvers for linear systems. Key applications of the HSS algorithms, coupled with the sparse matrix techniques such as the multifrontal solvers, have been shown very useful in solving certain large-scale discretized PDEs and computational inverse problems [25, 29]. For example, they can be built into parallel sparse solvers for Helmholtz equations arising from frequency domain wave equation modeling prevailing in the oil and gas industry. In particular, we point out an application to the multi-frequency formulation of (seismic) inverse scattering and tomography, where a Helmholtz equation has to be solved for many right-hand sides, on a large domain for a selected set of frequencies. The solutions are combined to compute one step in, for example, a nonlinear Landweber iteration. The computational accuracy can be controlled, namely, in concert with the accuracy of

*Center for Computational and Applied Mathematics, Purdue University, West Lafayette, IN 47907 (wang273@math.purdue.edu).

†Lawrence Berkeley National Laboratory, Berkeley, CA 94720 (xsli@lbl.gov). The research of this author was supported in part by the Director, Office of Science, Office of Advanced Scientific Computing Research of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231.

‡Center for Computational and Applied Mathematics, Purdue University, West Lafayette, IN 47907 (xiaj@math.purdue.edu). The research of this author was supported in part by NSF grants DMS-1115572 and CHE-0957024.

§Department of Computer Science, Purdue University, West Lafayette, IN 47907 (ysitu@cs.purdue.edu).

¶Center for Computational and Applied Mathematics, Purdue University, West Lafayette, IN 47907 (mdehoop@math.purdue.edu).

the data. The HSS methods can be used to obtain fast approximate direct solutions and have been shown very useful for such problems [25, 26].

An HSS representation has a nice binary tree structure, called the HSS tree, and the HSS operations can be generally conducted following the traversal of this tree in a parallel fashion. However, the existing studies of the HSS structures focus only on their mathematical aspects, and the current HSS methods are only implemented in sequential computations. Similar limitations also exist for some other rank structured methods.

Here, we present the new parallel and efficient HSS algorithms and study their scalability. We concentrate on three most significant HSS algorithms: The parallel construction of an HSS representation or approximation for a dense matrix using a parallel rank-revealing QR (RRQR) factorization, the parallel ULV-type factorization [7] of such a matrix, and the parallel solution. The complexity of the HSS construction, factorization, and solution algorithms are $\mathcal{O}(rn^2)$, $\mathcal{O}(r^2n)$, and $\mathcal{O}(rn)$, respectively, where $r$ is the maximum numerical rank and $n$ is the size of the dense matrix [7, 30]. Here, we further discuss the scalability of these algorithms. Since the HSS algorithms mostly consist of dense matrix kernels, we use `BLACS` [3] and `ScaLAPACK` [23] as the parallel building blocks for those kernels. We construct a *context* (sub-communicator) for each node of the HSS tree. We also exploit the governing 2D-block-cyclic data distribution scheme used in `ScaLAPACK` to achieve high performance.

Our parallel HSS construction consists of three phases: parallel RRQR factorization based on a Modified Gram-Schmidt (MGS) method with column pivoting, parallel row compression, and parallel column compression. The parallel HSS factorization involves the use of two children's contexts for a given parent context. The communication patterns are composed of intra-context and inter-context ones. Similar strategies are also applied to the HSS solution. In the presentation, some tree techniques for symmetric positive definite HSS matrices in [31] are generalized so as to efficiently handle nonsymmetric matrices in parallel.

Analysis of the communication costs in these procedures are presented. For example, in the HSS construction, the number of messages and the number of words transferred are $O(r\log^2 P+\log P)$ and $O(rn\log P+r^2\log^2 P+rn)$, respectively, where $P$ is the number of processes and the logarithm is in base two. In our numerical experiments, we confirm the accuracy and the weak scaling of the methods when they are used as kernels for solving large 2D and 3D Helmholtz problems. We also demonstrate their strong scaling for a large dense Toeplitz matrix. The results show that our algorithms achieve high performance when the system is scaled up to 6.4 billion unknowns.

The outline of the paper is as follows. In Section 2, we present an overview of HSS structures. The fundamental parallelization strategy and the performance model are introduced in Section 3, where we also briefly discuss our use of `BLACS` and `ScaLAPACK` to implement the high performance kernels. In Section 4, we present our parallel HSS construction framework. The parallel HSS factorization is described in Section 5. In Section 6, we discuss the parallel solution strategy. Some computational experiments are given in Section 7.

**2. Overview of the HSS structures.** We briefly summarize the key concepts of the HSS structures following the definitions and notation in [27, 30]. Let $A$ be a general $n\times n$ real or complex matrix and $\mathcal{I}=\{1,2,...,n\}$ be the set of all row and column indices. Suppose $\mathcal{T}$ is a postordered full binary tree with $2k-1$ nodes labeled as $i=1,2,\ldots,2k-1$, such that the root node is $2k-1$ and the number of leaf nodes

is $k$. That is, for each non-leaf node $i$ of $\mathcal{T}$, its left child $c_1$ and right child $c_2$ satisfy $c_1 < c_2 < i$. Let $t_i \subset \mathcal{I}$ be an index subset associated with each node $i$ of $\mathcal{T}$. We use $A|_{t_i \times t_j}$ to denote the submatrix of $A$ with row index subset $t_i$ and column index subset $t_j$. Then a postordered HSS form is defined as follows:

DEFINITION 2.1. *We say $A$ is in a postordered HSS form with the corresponding HSS tree $\mathcal{T}$ if the following conditions are satisfied:*

- *$t_{c_1} \cap t_{c_2} = \emptyset, t_{c_1} \cup t_{c_2} = t_i$ for each non-leaf node $i$ of $\mathcal{T}$ with children $c_1$ and $c_2$, and $t_{2k-1} = \mathcal{I}$.*
- *There exist matrices $D_i, U_i, R_i, B_i, W_i, V_i$ (called HSS generators) associated with each node $i$ of $\mathcal{T}$, such that*

$$\begin{aligned} &D_{2k-1} = A, \quad U_{2k-1} = \emptyset, \quad V_{2k-1} = \emptyset, \\ &D_i = A|_{t_i \times t_i} = \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^H \\ U_{c_2} B_{c_2} V_{c_1}^H & D_{c_2} \end{pmatrix}, \\ &U_i = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1} W_{c_1} \\ V_{c_2} W_{c_2} \end{pmatrix}, \end{aligned} \tag{2.1}$$

*where the superscript $H$ denotes the Hermitian transpose.*

The HSS generators define the HSS form of $A$. For each diagonal block $D_i = A|_{t_i \times t_i}$ associated with each node $i$ of $\mathcal{T}$, we define $A_i^- = A|_{t_i \times (\mathcal{I} \setminus t_i)}$ to be the *HSS block row*, and $A_i^| = A|_{(\mathcal{I} \setminus t_i) \times t_i}$ to be the *HSS block column*. They are both called *HSS blocks*. The maximum numerical rank $r$ of all the HSS blocks is called the *HSS rank* of $A$. If $r$ is small as compared with the matrix size, we say that $A$ has a *low-rank property*.

Figure 2.1 illustrates a block $8 \times 8$ HSS representation $A$. As a special example, its leading block $4 \times 4$ part looks like:

$$A|_{t_7 \times t_7} = \begin{pmatrix} \begin{pmatrix} D_1 & U_1 B_1 V_2^H \\ U_2 B_2 V_1^H & D_2 \end{pmatrix} & \begin{pmatrix} U_1 R_1 \\ U_2 R_2 \end{pmatrix} B_3 \begin{pmatrix} W_4^H V_4^H & W_5^H V_5^H \end{pmatrix} \\ \begin{pmatrix} U_4 R_4 \\ U_5 R_5 \end{pmatrix} B_6 \begin{pmatrix} W_1^H V_1^H & W_2^H V_2^H \end{pmatrix} & \begin{pmatrix} D_4 & U_4 B_4 V_5^H \\ U_5 B_5 V_4^H & D_5 \end{pmatrix} \end{pmatrix}.$$

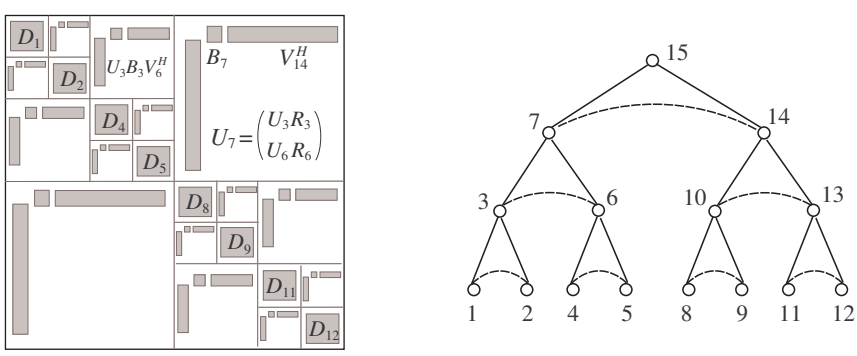



FIG. 2.1. *Pictorial illustrations of a block $8 \times 8$ HSS form and the corresponding HSS tree $\mathcal{T}$.*

Given a general dense matrix $A$ with the low-rank property, we can construct an HSS representation in parallel (or approximation when compression is used), upon which the HSS factorization and solution are conducted.

**3. Parallelization strategy.** For ease of exposition, we assume that all the diagonal blocks associated with the leaf nodes have the same block size $m$. We choose $m$ first, which is related to the HSS rank, and then choose the HSS tree and the number of processes $P \approx n/m$. For simplicity, assume $P$ is a power of two. Some existing serial HSS algorithms traverse the HSS tree in a postorder [27, 30]. For the HSS construction, the postordered traversal allows us to take advantage of previously compressed forms in later compression steps. However, the postordered HSS construction is serial in nature and involves global access of the matrix entries [30], and is not suitable for parallel computation. To facilitate parallelism, we reorganize the algorithms so that the HSS trees are traversed level by level. The complexity remains roughly the same. In fact, for HSS constructions, although the flop count with the levelwise traversal is slightly higher than with the postorder traversal, the leading terms in the counts are the same.

All the parallel operations are performed in either an upward sweep or a downward sweep along the HSS tree $\mathcal{T}$. We refer to the leaf/bottom level of the tree as level 1, and the next level up as level 2, and so on. We use the example in Figure 2.1 to illustrate the organization of the algorithms. The matrix is partitioned into eight block rows (Figure 3.1). We use eight processes $\{0, 1, 2, 3, 4, 5, 6, 7\}$ for the parallel operations. Each process individually works on one leaf node at level 1 of $\mathcal{T}$. At the second level, each group of two processes cooperate at a level-2 node. At the third level, each group of four processes cooperate at a level-3 node, and so on.

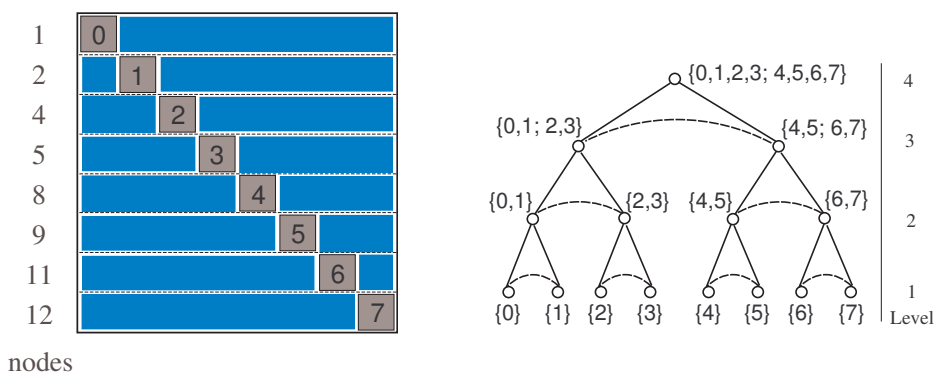



FIG. 3.1. *A block partition of a dense matrix A before the construction of the HSS form in Figure 2.1, where the labels inside the matrix and beside the nodes show the processes assigned to the nodes of the HSS tree in Figure 2.1.*

**3.1. Using `ScaLAPACK` and `BLACS`.** Since the HSS algorithms mostly consist of dense matrix kernels, we chose to use where possible the well established routines in the `ScaLAPACK` library [23] and its communication substrate, the `BLACS` library [3]. The governing distribution scheme is a 2D block cyclic matrix layout, in which the user specifies the block size of a submatrix and the shape of the 2D process grid. The blocks of the matrices are then cyclically mapped to the process grid in both row and column dimensions. Furthermore, the processes can be divided into subgroups to work on independent parts of the calculations. Each subgroup is called a *context* in `BLACS` term, similar to the sub-communicator concept in MPI [22]. All our algorithms start with a global context created from the entire communicator, i.e., `MPI_COMM_WORLD`. When we move up the HSS tree, we define the other contexts encompassing process

subgroups.

For example, in the example shown in Figures 2.1 and 3.1, the eight processes can be arranged as eight contexts for the leaf nodes in $\mathcal{T}$. Use $\{0,1\} \leftrightarrow$ node 3 to denote a context, which means that the set of processes $\{0,1\}$ is mapped to node 3. Four contexts are defined at the second level:

$$\{0,1\} \leftrightarrow \text{node } 3,\ \{2,3\} \leftrightarrow \text{node } 6,\ \{4,5\} \leftrightarrow \text{node } 10, \text{ and } \{6,7\} \leftrightarrow \text{node } 13.$$

Two contexts are defined at the third level:

$$\{0,1;\ 2,3\} \leftrightarrow \text{node } 7,\ \{4,5;\ 6,7\} \leftrightarrow \text{node } 14,$$

where the notation $\{0,1;\ 2,3\}$ means that processes 0 and 1 are stacked atop processes 2 and 3. Finally, one context is defined:

$$[0,1,4,5;\ 2,3,6,7] \leftrightarrow \text{node } 15.$$

We always arrange the process grid as square as possible, i.e., $P \approx \sqrt{P} \times \sqrt{P}$, and we can conveniently use $\sqrt{P}$ to refer to the number of processes in the row or column dimension.

When the algorithms move up the HSS tree, we need to perform a *redistribution* to merge the data distributed in the two children's process contexts to the parent's context. Since the two children's contexts have about the same size and shape (due to the low-rank property) and the parent context roughly doubles each child's context, the parent context can be arranged to combine the two children's contexts either side by side or one on top of the other. Thus, the processes grid is kept as square as possible, and the redistribution pattern is simple, which only involves *pairwise exchanges*. That is, only the pair of processes at the same coordinate in the two children's contexts exchange data. For example, the redistribution from $\{0,1;\ 2,3\} + \{4,5;\ 6,7\}$ to $\{0,1,4,5;\ 2,3,6,7\}$ is achieved by the following pairwise exchanges: $0 \leftrightarrow 4$, $1 \leftrightarrow 5$, $2 \leftrightarrow 6$, and $3 \leftrightarrow 7$.

**3.2. Parallel performance model.** We will use the following notation in the analysis of the communication cost of our parallel algorithms:

- $r$ is the HSS rank of $A$.
- The pair [`#messages`, `#words`] is used to count the number of messages and the number of words transferred. The parallel runtime can be modeled as the following (ignoring the overlap of communication with computation):

$$\texttt{Time} = \texttt{\#flops} \cdot \gamma + \texttt{\#messages} \cdot \alpha + \texttt{\#words} \cdot \beta\ ,$$

  where $\gamma$, $\alpha$, and $\beta$ are the time taken for each flop, each message (latency), and each word transferred (reciprocal bandwidth), respectively.
- The cost of broadcasting a message of $W$ words among $P$ processes is modeled as $[\log P,\ W \log P]$, assuming a tree-based or hypercube-based broadcast algorithm is used. The same cost is incurred for a reduction operation of $W$ words.

**4. Parallel HSS construction.** In this section we discuss the construction of an HSS representation (or approximation) for $A$ in parallel. Unlike the methods in [28, 30], our discussions focus on the computational aspects and the performance scalability. The construction is composed of a row compression step (Section 4.2) followed by a column compression step (Section 4.3). The key kernel is a parallel RRQR algorithm which we discuss first.

**4.1. Parallel RRQR factorization or compression.** The key step in the HSS construction is to compress the HSS blocks of $A$. For example, consider an HSS block $F$. Truncated SVD is one option to realize such compression. That is, we drop those singular values below a prescribed threshold of all the singular values of $F$. SVD is generally very expensive. An efficient alternative is to use rank-revealing QR (RRQR), where QR factorization with column pivoting is performed. We now describe our parallel RRQR algorithm.

For notational simplicity, assume the numerical rank (determined by a given relative tolerance $\tau$) is equal to the HSS rank $r$. Assume $F$ is of size $M \times N$, and is distributed in the process context $P \approx \sqrt{P} \times \sqrt{P}$. That is, the local dimension of $F$ is $\frac{M}{\sqrt{P}} \times \frac{N}{\sqrt{P}}$. The following algorithm, based on a Modified Gram-Schmidt strategy which is revised from a RRQR factorization scheme in [13, 14], computes RRQR in parallel: $F \approx \widetilde{Q}\widetilde{T}$ where $\widetilde{Q} = (q_1, q_2, ..., q_r)$ and $\widetilde{T}^H = (t_1, t_2, ..., t_r)$. See Table 4.1.

```
subroutine [Q,T] ≈ RRQR(F,τ)
for i = 1:r
    1.  In parallel, find the column f_j of F with the maximum norm
    2.  Interchange f_i and f_j
    3.  Compute t_ii = ||f_i||, and if t_ii/t_11 ≤ τ, stop
    4.  Normalize f_i:  q_i = f_i/||f_i||
    5.  Broadcast q_i row-wise within the context in which F resides
    6.  PBLAS2:  t_j^H = q_i^H (f_{i+1}, t_{i+2}, ..., f_N)
    7.  Compute rank-1 update: (f_{i+1}, t_{i+2}, ..., f_N) = (f_{i+1}, t_{i+2}, ..., f_N) − q_j t_j^H
end
```

TABLE 4.1

*Parallel RRQR factorization of $F$ with a relative tolerance $\tau$, where the norm is 2-norm. (Note that Step 2 is used for the clarity of explanation and is not done in actual computations. Also Step 3 is done quickly by the norm update strategy as in [13].)*

Communications occur in Steps 1 and 5. The other steps only involve local computations. In Step 1, the processes in each column group perform one reduction of size $\frac{N}{\sqrt{P}}$ to compute the column norms, with communication cost $[\log\sqrt{P},\ \frac{N}{\sqrt{P}}\log\sqrt{P}]$. This is followed by another reduction among the processes in the row group to find the maximum norm among all the columns, with communication cost $[\log\sqrt{P},\ \log\sqrt{P}]$. In Step 3, the processes among each row group broadcast $q_j$ of size $\frac{M}{\sqrt{P}}$, costing $[\log\sqrt{P},\ \frac{M}{\sqrt{P}}\log\sqrt{P}]$.

Summing the leading terms for $r$ steps, we obtain the following communication cost:

$$\texttt{Comm}_{RRQR} = \left[\log\sqrt{P},\ \frac{M+N}{\sqrt{P}}\log\sqrt{P}\right]\cdot r\ . \tag{4.1}$$

To achieve higher performance, a block RRQR strategy can be adopted similarly, like the `LAPACK` subroutine `xGEQP3` [20].

**4.2. Parallel row compression stage.** We still use the block $8 \times 8$ matrix in Figures 2.1 and 3.1 to illustrate the algorithm step by step.

**4.2.1. Row compression – level 1.** In the first step, all the leaves 1, 2, 4, 5, 8, 9, 11, and 12 of $\mathcal{T}$ have their own processes $\{0\}, \{1\}, \{2\}, \{3\}, \{4\}, \{5\}, \{6\}$, and $\{7\}$, respectively. Each process owns part of the global matrix $A$, given by $D_i = A|_{t_i \times t_i}$

and $A_i^- = A|_{t_i \times (\mathcal{I} \setminus t_i)}$, as illustrated in Figure 3.1. The block to be compressed is $F_i \equiv A_i^-$. We perform a parallel RRQR factorization (Section 4.1) on $F_i$:

$$F_i \approx U_i \widehat{F}_i.$$

For notational convenience, we write $\widehat{F}_i \equiv A|_{\hat{t}_i \times (\mathcal{I} \setminus t_i)}$, which can be understood as that $\widehat{F}_i$ is stored in the space of $F_i$, or in $A$ with row index set $\hat{t}_i$. That is, we can write the above factorization as[1]

$$A|_{t_i \times (\mathcal{I} \setminus t_i)} \approx U_i A|_{\hat{t}_i \times (\mathcal{I} \setminus t_i)}. \tag{4.2}$$

This step is done locally within each process. One of the HSS generators $U_i$ is obtained here.

We also prepare for the compression at the upper level, level 2 of $\mathcal{T}$. The upper level compression must be carried out among a pair of processes in each context at level 1. For this purpose, we need a *redistribution* phase prior to the compression. That is, we perform pairwise exchange of data: $\{0\} \leftrightarrow \{1\}$, $\{2\} \leftrightarrow \{3\}$, $\{4\} \leftrightarrow \{5\}$, and $\{6\} \leftrightarrow \{7\}$. The level-2 nodes on $\mathcal{T}$ are 3, 6, 10, and 13, whose contexts are $\{0,1\}$, $\{2,3\}$, $\{4,5\}$, and $\{6,7\}$ respectively. For each node $i$ at level 2 with children $c_1$ and $c_2$ (at level 1), we have

$$A|_{t_{c_1} \times t_{c_2}} \approx U_{c_1} A|_{\hat{t}_{c_1} \times t_{c_2}}, \; A|_{t_{c_2} \times t_{c_1}} \approx U_{c_2} A|_{\hat{t}_{c_2} \times t_{c_1}}, \; A_i^- \approx \begin{pmatrix} U_{c_1} A|_{\hat{t}_{c_1} \times (\mathcal{I} \setminus t_i)} \\ U_{c_2} A|_{\hat{t}_{c_2} \times (\mathcal{I} \setminus t_i)} \end{pmatrix}.$$

Ignoring the basis matrices $U_{c_1}$ and $U_{c_2}$, the block to be compressed in the next step is

$$F_i \equiv \begin{pmatrix} A|_{\hat{t}_{c_1} \times (\mathcal{I} \setminus t_i)} \\ A|_{\hat{t}_{c_2} \times (\mathcal{I} \setminus t_i)} \end{pmatrix}. \tag{4.3}$$

This procedure is illustrated in Figure 4.1(a). Two communications steps are used. First, exchange $A|_{\hat{t}_{c_1} \times t_{c_2}}$ and $A|_{\hat{t}_{c_2} \times t_{c_1}}$ between $c_1$'s and $c_2$'s contexts. This prepares for the column compression later. Next, redistribute the newly merged off-diagonal block $F_i$ onto the process grid associated with the contexts $\{0,1\}$, $\{2,3\}$, $\{4,5\}$, and $\{6,7\}$ corresponding to nodes 3, 6, 10, and 13, respectively. Here we use a `ScaLAPACK` subroutine `PxGEMR2D` to realize the data exchange and redistribution steps.

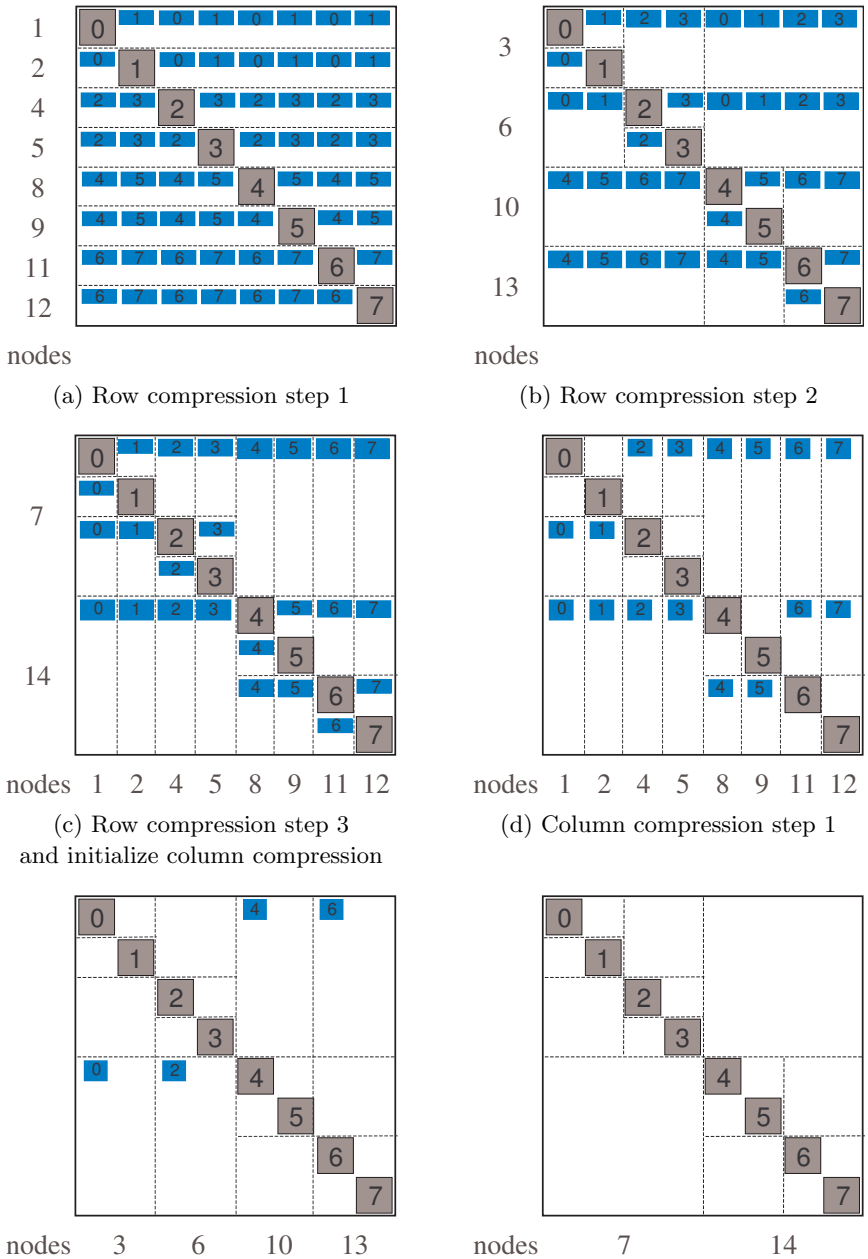
During the redistribution phase, the number of messages is 2, and the number of words exchanged is $\frac{rn}{2} \cdot 2$. The communication cost is $[2, \; \frac{rn}{2} \cdot 2]$.

**4.2.2. Row compression – level 2.** At level 2 of $\mathcal{T}$, within the context for each node $i = 3, 6, 10, 13$, we perform a parallel RRQR factorization for $F_i$ in (4.3):

$$F_i \approx \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} A|_{\hat{t}_i \times (\mathcal{I} \setminus t_i)}, \tag{4.4}$$

where $A|_{\hat{t}_i \times (\mathcal{I} \setminus t_i)}$ is defined similar to the one in (4.2). The $R$ generators associated with the child level are then obtained. Since the size of each $F_i$ is bounded by $2r \times n$ and two processes are used for each RRQR factorization, we obtain the communication cost $[\log \sqrt{2}, \; \frac{2r+n}{\sqrt{2}} \log \sqrt{2}] \cdot r$ using (4.1).

[1] This is only a way to simplify notation in the presentation, and is not the actual storage used in our implementation.




(a) Row compression step 1

(b) Row compression step 2

(c) Row compression step 3 and initialize column compression

(d) Column compression step 1

(e) Column compression step 2

(f) Column compression step 3

FIG. 4.1. *Illustration of data distribution in the row and column compressions, where the labels inside the matrix mean processes, and those outside mean the nodes of* $\mathcal{T}$.

To prepare for the compression at the upper level (level 3) of $\mathcal{T}$, again, we need a *redistribution* phase, performing the following pairwise data exchange: $\{0,1\} \leftrightarrow \{2,3\}$ and $\{4,5\} \leftrightarrow \{6,7\}$. In this notation, the exchanges $0 \leftrightarrow 1$ and $2 \leftrightarrow 3$ occur simultaneously. There is no need for data exchanges between processes 0 and 3, or 1 and 2. The level-3 nodes are 7 and 14 with the process contexts $\{0,1;\ 2,3\}$ and

$\{4,5;\ 6,7\}$, respectively. There are also communications similar to the procedure for forming (4.3) in the previous step, so that each of the two off-diagonal blocks $F_i,\ i = 7, 14$ is formed and distributed onto the respective process context. This is illustrated in Figure 4.1(b).

During the redistribution phase, the number of messages is 2, and the number of words exchanged is $\frac{rn}{2^2} \cdot 2$. Thus the communication cost is $[2,\ \frac{rn}{2^2} \cdot 2]$.

**4.2.3. Row compression – level 3.** At level 3, we perform the parallel RRQR within each context for each $F_i,\ i = 7, 14$, similar to (4.4). The $R$ generators associated with the child level are also obtained here. The communication cost of RRQR is given by $[\log\sqrt{4},\ \frac{2r+n}{\sqrt{4}}\log\sqrt{4}] \cdot r$.

Since the upper level node has only one node, the root node 15 of $\mathcal{T}$, there is no off-diagonal block associated with it. Thus, to prepare for the column HSS constructions, only one pairwise exchange step is needed between the two contexts: $\{0,1;\ 2,3\} \leftrightarrow \{4,5;\ 6,7\}$, meaning $0 \leftrightarrow 4, 1 \leftrightarrow 5, 2 \leftrightarrow 6$, and $3 \leftrightarrow 7$. This is similar to (4.3) except that there is no merging step to form $F_{15}$. The procedure is illustrated in Figure 4.1(c). The communication cost in the redistribution phase is $[2,\ \frac{rn}{2^3} \cdot 2]$.

**4.2.4. General case and summation of communication costs in row compression.** In general, the compression and communication for an HSS matrix with more blocks can be similarly shown. Here, we sum the messages and number of words communicated at all the levels of the tree in this row compression stage. For simplicity, assume there are $P$ leaves and about $L \approx \log P$ levels in $\mathcal{T}$. Then the total communication cost is summed up by the following:[2]

(1) Redistributions:

$$\texttt{\#messages} = \sum_{i=1}^{L} 2 \approx 2\log P,$$

$$\texttt{\#words} = \sum_{i=1}^{L} (\frac{rn}{2^i}2) = \mathcal{O}(2rn).$$

(2) RRQR factorizations:

$$\texttt{\#messages} = \sum_{i=1}^{L} (\log\sqrt{2^i}) \cdot r = r\sum_{i=1}^{L} \frac{i}{2} = \mathcal{O}(r\log^2 P),$$

$$\texttt{\#words} = \sum_{i=1}^{L} (\frac{2r+n}{\sqrt{2^i}}\log\sqrt{2^i}) \cdot r = r\left(\frac{2r+n}{2}\right)\sum_{i=1}^{L} \frac{i}{2^{i/2}} = \mathcal{O}(rn\log P).$$

**4.3. Parallel column compression stage.** After the row compression, the blocks $A|_{\hat{t}_j \times (\mathcal{I}\setminus t_j)}$ remained to be compressed in the column compression stage are much smaller. In addition, in this stage, pieces of the blocks $A|_{\hat{t}_j \times (\mathcal{I}\setminus t_j)}$ for nodes $j$ at different levels may be compressed together to get a $V$ generator. To illustrate this, we use the following definition, which generalizes the concept of visited sets in [31] for symmetric positive definite matrices to nonsymmetric ones.

Definition 4.1. *The* left visited set *associated with a node $i$ of a postordered full binary tree $\mathcal{T}$ is*

$$\mathcal{V}_i = \{j \mid j \textit{ is a left node and } \mathrm{sib}(j) \in \mathrm{ances}(i)\},$$

[2] In most situations that we are interested, we can assume $n \gg r$.

*where* $\mathrm{sib}(j)$ *is the sibling of* $i$ *in* $\mathcal{T}$ *and* $\mathrm{ances}(i)$ *is the set of ancestors of node* $i$ *including* $i$*. Similarly, the* right visited set *associated with* $i$ *is*

$$\mathcal{W}_i = \{j \mid j \textit{ is a right node and } \mathrm{sib}(j) \in \mathrm{ances}(i)\}.$$

$\mathcal{V}_i$ and $\mathcal{W}_i$ are essentially the stacks before the visit of $i$ in the postordered and reverse-postordered traversals of $\mathcal{T}$, respectively.

We now describe how the column compression works. We still use the same $8 \times 8$ block matrix example after the row compression for illustration.

**4.3.1. Column compression − level 1.** After the row compression, the updated off-diagonal blocks $\widehat{F}_j \equiv A|_{\hat{t}_j \times (\mathcal{I} \setminus t_j)}$, $j = 1, 2, \ldots, 14$ are stored in the individual contexts, at different levels of the HSS tree. For example, $F_1$ is stored in the context $\{0\}$, $F_3$ is stored in the context $\{0, 1\}$, and $F_7$ is stored in the context $\{0, 1;\ 2, 3\}$. The algorithm for the column compression is also in an upward sweeping along $\mathcal{T}$ [30]. To prepare for the compression associated with the leaf nodes, we first need a redistribution phase to transfer the $A|_{\hat{t}_j \times \mathrm{sib}(t_j)}$ blocks for nodes $j$ at the higher levels to the bottom leaf level. This is achieved in $\log P$ steps of communication in a top-down fashion. In each step, we redistribute $A|_{\hat{t}_j \times \mathrm{sib}(t_j)}$ in the context of $j$ to the contexts of the leaf nodes. These $j$ indices of the row blocks that need to be redistributed downward are precisely those in the visited sets in Definition 4.1. For instance, the leaf node 2 needs the pieces corresponding to the nodes $\mathcal{V}_2 \cup \mathcal{W}_2 = \{1\} \cup \{6, 14\}$. For all the leaf nodes, the redistribution procedure achieves the following blocks which need to be compressed in this stage:

$$G_1^H = \begin{pmatrix} A|_{\hat{t}_2 \times t_1} \\ A|_{\hat{t}_6 \times t_1} \\ A|_{\hat{t}_{14} \times t_1} \end{pmatrix},\ G_2^H = \begin{pmatrix} A|_{\hat{t}_1 \times t_2} \\ A|_{\hat{t}_6 \times t_2} \\ A|_{\hat{t}_{14} \times t_2} \end{pmatrix},\ G_4^H = \begin{pmatrix} A|_{\hat{t}_3 \times t_4} \\ A|_{\hat{t}_5 \times t_4} \\ A|_{\hat{t}_{14} \times t_4} \end{pmatrix},\ G_5^H = \begin{pmatrix} A|_{\hat{t}_4 \times t_5} \\ A|_{\hat{t}_3 \times t_5} \\ A|_{\hat{t}_{14} \times t_5} \end{pmatrix},$$
$$G_8^H = \begin{pmatrix} A|_{\hat{t}_9 \times t_8} \\ A|_{\hat{t}_{13} \times t_8} \\ A|_{\hat{t}_7 \times t_8} \end{pmatrix},\ G_9^H = \begin{pmatrix} A|_{\hat{t}_8 \times t_9} \\ A|_{\hat{t}_{13} \times t_9} \\ A|_{\hat{t}_7 \times t_9} \end{pmatrix},\ G_{11}^H = \begin{pmatrix} A|_{\hat{t}_{12} \times t_{11}} \\ A|_{\hat{t}_{10} \times t_{11}} \\ A|_{\hat{t}_7 \times t_{11}} \end{pmatrix},\ G_{12}^H = \begin{pmatrix} A|_{\hat{t}_{11} \times t_{12}} \\ A|_{\hat{t}_{10} \times t_{12}} \\ A|_{\hat{t}_7 \times t_{12}} \end{pmatrix}. \tag{4.5}$$

We can use $\mathcal{V}_i$ and $\mathcal{W}_i$ to simplify the notation. For example, we write

$$\bar{t}_1 = \hat{t}_2 \cup \hat{t}_6 \cup \hat{t}_{14},\ G_1^H = A|_{\bar{t}_1 \times t_1}.$$

We still use the `ScaLAPACK` subroutine `PxGEMR2D` to perform these inter-context communications. In this redistribution phase, the number of messages sent is $\log P$, and the number of words is $\frac{r\ n}{\sqrt{P}}$. Thus the communication cost is $[\log P,\ \frac{rn}{\sqrt{P}} \log P]$.

After the redistribution, the layout of the off-diagonal blocks is illustrated by Figure 4.1(c), which initiates the parallel column construction. At the bottom level, the contexts $\{0\}$, $\{1\}$, $\{2\}$, $\{3\}$, $\{4\}$, $\{5\}$, $\{6\}$, and $\{7\}$ are associated with the leaf nodes 1, 2, 4, 5, 8, 9, 11, and 12, respectively. $G_i^H$ for all leaves $i$ are indicated by the shaded areas in Figure 4.1(c). We carry out a parallel RRQR factorization on $G_i$:

$$G_i \approx V_i \widetilde{G}_i.$$

This can be denoted as

$$G_i \approx A|_{\bar{t}_i \times \tilde{t}_i} V_i^H,$$

where $\tilde{t}_i$ is a subset of $t_i$ and we can understand as that $\widetilde{G}_i = \left(A|_{\bar{t}_i \times \tilde{t}_i}\right)^H$ can be stored in the space of $G_i$. (This is solely for notational convenience. See the remark for (4.2).) We note that this step is done locally within each process. The $V$ generators are obtained here.

To prepare for the upper level column compression, communications occur pairwise: $\{0\} \leftrightarrow \{1\}$, $\{2\} \leftrightarrow \{3\}$, $\{4\} \leftrightarrow \{5\}$, $\{6\} \leftrightarrow \{7\}$. The upper level blocks $G_i,\ i = 3, 6, 10, 13$ for compression are formed by ignoring the $V$ basis matrices and merging parts of $A|_{\bar{t}_{c_1} \times \tilde{t}_{c_1}}$ and $A|_{\bar{t}_{c_2} \times \tilde{t}_{c_2}}$. That is, we set

$$G_i = \left( \begin{array}{cc} A|_{\bar{t}_i \times \tilde{t}_{c_1}}, & A|_{\bar{t}_i \times \tilde{t}_{c_2}} \end{array} \right),\ B_{c_1} = A|_{\hat{t}_{c_1} \times \tilde{t}_{c_2}},\ B_{c_2} = A|_{\hat{t}_{c_2} \times \tilde{t}_{c_1}}. \tag{4.6}$$

This procedure is illustrated in Figure 4.1(d). Two communication steps are needed. In the first step $B_{c_1}$ and $B_{c_2}$ are generated by exchanging $A|_{\hat{t}_{c_2} \times \tilde{t}_{c_1}}$ and $A|_{\hat{t}_{c_1} \times \tilde{t}_{c_2}}$ pairwise between $c_1$'s and $c_2$'s contexts. We note that some $B$ generators are obtained here. The second step is to redistribute the newly merged off-diagonal block $G_i$ onto the process grid associated with the contexts for nodes $i = 3, 6, 10, 13$.

We note that during the column compression stage, the number of nodes in $\mathcal{V}_i \cup \mathcal{W}_i$ needed to form $G_i$ is the same as the number of levels in the HSS tree, which is $\log(\frac{n}{m}) \approx P$. See, e.g., (4.5). Therefore, the row dimension of $G_i$ is bounded by $r \log(\frac{n}{m})$, which is much smaller than the column dimension $n$ during the row compression stage. Similar to the level 1 row compression, during this redistribution phase, the number of messages is 2 and the number of words exchanged is $\frac{r^2 \log P}{2} \cdot 2$. The communication cost is then $[2,\ \frac{r^2 \log P}{2} \cdot 2]$.

**4.3.2. Column compression – level 2.** At level 2, the contexts $\{0, 1\}$, $\{2, 3\}$, $\{4, 5\}$, and $\{6, 7\}$ are associated with the nodes 3, 6, 10, and 13, respectively. Each off-diagonal block $G_i,\ i = 3, 6, 10, 13$ has already been distributed onto the respective process context, as illustrated in Figure 4.1(d). Then we perform a parallel RRQR factorization on each $G_i$:

$$G_i = \begin{pmatrix} W_{c_1} \\ W_{c_2} \end{pmatrix} \widetilde{G}_i,\ \widetilde{G}_i \equiv \left(A|_{\bar{t}_i \times \tilde{t}_i}\right)^T. \tag{4.7}$$

Some $W$ generators are obtained. Since each $G_i$ is bounded by the size $r \log P \times 2r$ and two processes are used for each RRQR factorization, using (4.1), we obtain the communication cost $[\log \sqrt{2},\ \frac{2r + r \log P}{\sqrt{2}} \log \sqrt{2}] \cdot r$.

To enable the upper level column HSS construction, communication occurs pairwise: $\{0, 1\} \leftrightarrow \{2, 3\}$ and $\{4, 5\} \leftrightarrow \{6, 7\}$. The procedure is illustrated by Figure 4.1(e). Similar to (4.6), two communication steps are needed. During the distribution phase, the number of messages is 2, and the number of words exchanged is $\frac{r^2 \log P}{4} \cdot 2$. The communication cost is $[2,\ \frac{r^2 \log P}{4} \cdot 2]$.

**4.3.3. Column compression – level 3.** At level 3, the two contexts $\{0, 1;\ 2, 3\}$ and $\{4, 5;\ 6, 7\}$ are associated with the nodes 7 and 14, respectively. Each off-diagonal block $F_i,\ i = 7, 14$ has already been distributed onto the respective process contexts, as shown in Figure 4.1(e). Then we perform RRQR factorizations similarly to (4.7). The communication cost of RRQR is given by $[\log \sqrt{4},\ \frac{2r + r \log P}{\sqrt{4}} \log \sqrt{4}] \cdot r$.

Since the level-4 node is the root node 15 of $\mathcal{T}$, there is no off-diagonal block $F_{15}$ associated with it. Thus, the entire parallel HSS construction is finalized at this step. There is only one stage of communications occurring: $\{0, 1;\ 2, 3\} \leftrightarrow \{4, 5;\ 6, 7\}$, which

is similar to (4.6) except there is no merging step. Figure 4.1(f) indicates that after this final communication, all the HSS generators are obtained. The communication cost is $[2, \frac{r^2 \log P}{8} \cdot 2]$.

**4.3.4. Summation of communication costs in column compression.** We now sum all the messages and words communicated at all the levels of the tree during the column compression, and obtain the total communication costs as follows, where $L \approx \log P$:

(1) Redistributions:

$$\texttt{\#messages} = \log P + \sum_{i=1}^{L} 2 \approx 3 \log P,$$

$$\texttt{\#words} = \frac{rn}{\sqrt{P}} \log P + \sum_{i=1}^{L} (\frac{r^2 \log P}{2^i} 2) = \mathcal{O}(rn).$$

(2) RRQR factorizations:

$$\texttt{\#messages} = \sum_{i=1}^{L} \log \sqrt{2^i} \cdot r = r \sum_{i=1}^{L} \frac{i}{2} = \mathcal{O}(r \log^2 P),$$

$$\begin{aligned}\texttt{\#words} &= \sum_{i=1}^{L} \left( \frac{2r + r \log P}{\sqrt{2^i}} \log \sqrt{2^i} \right) \cdot r \\ &= r \cdot \frac{2r + r \log P}{2} \sum_{i=1}^{L} \frac{i}{2^{i/2}} = \mathcal{O}(r^2 \log^2 P).\end{aligned}$$

**4.4. Total communication cost in parallel HSS construction.** After the two stages of compression, all the HSS generators $D_i, U_i, R_i, B_i, W_i, V_i$ are obtained. The following formulas summarize the total communication costs for the entire parallel HSS construction, including both the row construction and the column construction:

(1) Redistributions:

$$\texttt{\#messages} = \mathcal{O}(\log P), \tag{4.8}$$

$$\texttt{\#words} = \mathcal{O}(rn). \tag{4.9}$$

(2) RRQR factorizations:

$$\texttt{\#messages} = \mathcal{O}(r \log^2 P), \tag{4.10}$$

$$\texttt{\#words} = \mathcal{O}(rn \log P + r^2 \log^2 P). \tag{4.11}$$

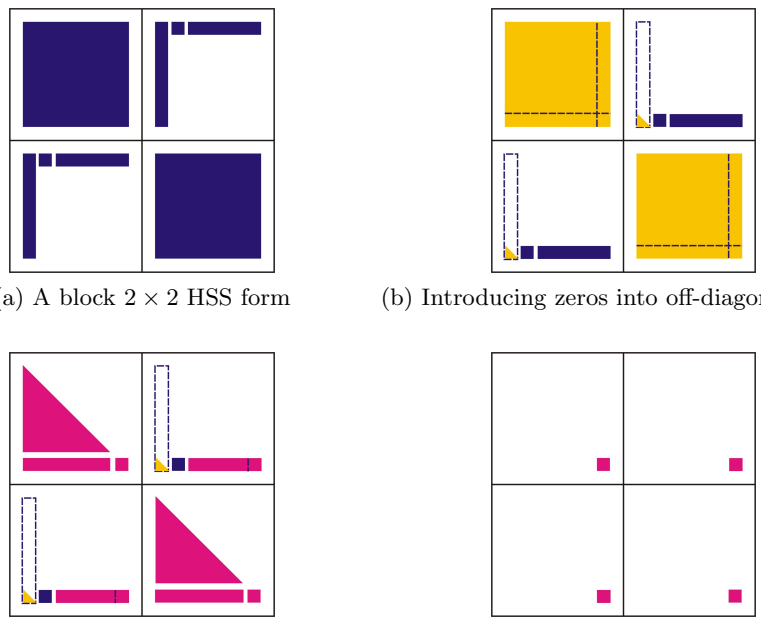
Comparing (4.8)–(4.11), we see that RRQR factorizations dominate the communication costs both in message count and in message volume. This is validated in our performance tests in Section 7.

Putting this in perspective, we compare the communication complexity to the flop count. It was shown earlier that the total `#flops` in this phase is $\mathcal{O}(rn^2)$ [30]. Then, given a perfect load balance, the flop count per process is $\mathcal{O}(\frac{rn^2}{P})$. Taking (4.11) to be the dominant communication part, the *flop-to-byte ratio* is roughly $\frac{n}{P \log P}$, which is very small. This indicates that our parallel algorithm is very much communication bound, and its parallel performance is more sensitive to the network speed than the CPU speed.

**5. Parallel ULV HSS factorization.** After the HSS approximation to $A$ is constructed, we are ready to factorize it via the generators. Here we adopt the ULV-type factorization method in [7] and present our parallel strategy in terms of a block $2 \times 2$ HSS form (Figure 5.1(a)):

$$\begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^H \\ U_{c_2} B_{c_2} V_{c_1}^H & D_{c_2} \end{pmatrix}, \tag{5.1}$$

where $c_1$ and $c_2$ are children of a node $i$ and are leaves of the HSS tree $\mathcal{T}$, and the generators associated with $c_1$ and $c_2$ are distributed on the process grids corresponding to the contexts of $c_1$ and $c_2$, respectively. The context of $i$ is the union of the contexts of $c_1$ and $c_2$. We assume that the sizes of $U_{c_1}$ and $U_{c_2}$ are $m \times r$.



(a) A block $2 \times 2$ HSS form

(b) Introducing zeros into off-diagonal blocks

(c) Introducing zeros into diagonal blocks

(d) After partial diagonal elimination

FIG. 5.1. *(a). The ULV factorization of a block $2 \times 2$ HSS form and the illustration of the inter-context communication to form (5.4).*

We start with the QL factorization of $U_{c_1}$ and $U_{c_2}$:

$$U_{c_1} = Q_{c_1} \begin{pmatrix} 0 \\ \widetilde{U}_{c_1} \end{pmatrix}, \quad U_{c_2} = Q_{c_2} \begin{pmatrix} 0 \\ \widetilde{U}_{c_2} \end{pmatrix}, \tag{5.2}$$

where $\widetilde{U}_{c_1}$ and $\widetilde{U}_{c_2}$ are lower triangular matrices of size $r \times r$, respectively. (In fact, since $U_{c_1}$ and $U_{c_2}$ have orthonormal columns in our HSS construction, we can directly derive orthogonal matrices them so that $\widetilde{U}_{c_1}$ and $\widetilde{U}_{c_2}$ become identity matrices.) We note that there is no inter-context communication at this stage. We multiply $Q_{c_1}^H$ and $Q_{c_2}^H$ to the block rows independently within each context and obtain

$$\begin{pmatrix} Q_{c_1}^H & 0 \\ 0 & Q_{c_2}^H \end{pmatrix} \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^H \\ U_{c_2} B_{c_2} V_{c_1}^H & D_{c_2} \end{pmatrix} = \begin{pmatrix} \widehat{D}_{c_1} & \begin{pmatrix} 0 \\ \widetilde{U}_{c_1} \end{pmatrix} B_{c_1} V_{c_2}^H \\ \begin{pmatrix} 0 \\ \widetilde{U}_{c_2} \end{pmatrix} B_{c_2} V_{c_1}^H & \widehat{D}_{c_2} \end{pmatrix}.$$

This is illustrated by Figure 5.1(b).

Next, we partition the diagonal blocks conformably as

$$\widehat{D}_{c_1} = Q_{c_1}^H D_{c_1} = \begin{pmatrix} \widehat{D}_{c_1;1,1} & \widehat{D}_{c_1;1,2} \\ \widehat{D}_{c_1;2,1} & \widehat{D}_{c_1;2,2} \end{pmatrix}, \quad \widehat{D}_{c_2} = Q_{c_2}^H D_{c_2} = \begin{pmatrix} \widehat{D}_{c_2;1,1} & \widehat{D}_{c_2;1,2} \\ \widehat{D}_{c_2;2,1} & \widehat{D}_{c_2;2,2} \end{pmatrix},$$

where $\widehat{D}_{c_1;2,2}$ and $\widehat{D}_{c_2;2,2}$ are of size $r \times r$, respectively. Compute an LQ factorization independently within each context:

$$\begin{pmatrix} \widehat{D}_{c_1;1,1} & \widehat{D}_{c_1;1,2} \end{pmatrix} = \begin{pmatrix} \widetilde{D}_{c_1;1,1} & 0 \end{pmatrix} P_{c_1},$$
$$\begin{pmatrix} \widehat{D}_{c_2;1,1} & \widehat{D}_{c_2;1,2} \end{pmatrix} = \begin{pmatrix} \widetilde{D}_{c_2;1,1} & 0 \end{pmatrix} P_{c_2}.$$

We multiply $P_{c_1}$ and $P_{c_2}$ to the block columns independently within each context and obtain:

$$\begin{pmatrix} Q_{c_1}^H & 0 \\ 0 & Q_{c_2}^H \end{pmatrix} \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^H \\ U_{c_2} B_{c_2} V_{c_1}^H & D_{c_2} \end{pmatrix} \begin{pmatrix} P_{c_1}^H & 0 \\ 0 & P_{c_2}^H \end{pmatrix}$$
$$= \begin{pmatrix} \begin{pmatrix} \widetilde{D}_{c_1;1,1} & 0 \\ \widetilde{D}_{c_1;2,1} & \widetilde{D}_{c_1;2,2} \end{pmatrix} & \begin{pmatrix} 0 \\ \widetilde{U}_{c_1} B_{c_1} \begin{pmatrix} \widetilde{V}_{c_2;1}^H & \widetilde{V}_{c_2;2}^H \end{pmatrix} \end{pmatrix} \\ \begin{pmatrix} 0 \\ \widetilde{U}_{c_2} B_{c_2} \begin{pmatrix} \widetilde{V}_{c_1;1}^H & \widetilde{V}_{c_1;2}^H \end{pmatrix} \end{pmatrix} & \begin{pmatrix} \widetilde{D}_{c_2;1,1} & 0 \\ \widetilde{D}_{c_2;2,1} & \widetilde{D}_{c_2;2,2} \end{pmatrix} \end{pmatrix}, \tag{5.3}$$

where the blocks are partitioned conformably. See Figure 5.1(c). We note that there is still no inter-context communication up to this stage.

Then we remove $c_1$ and $c_2$ form $\mathcal{T}$, and $i$ becomes a leaf, which we assign new generators:

$$D_i = \begin{pmatrix} \widetilde{D}_{c_1;2,2} & \widetilde{U}_{c_1} B_{c_1} \widetilde{V}_{c_2;2}^H \\ \widetilde{U}_{c_2} B_{c_2} \widetilde{V}_{c_1;2}^H & \widetilde{D}_{c_2;2,2} \end{pmatrix}, \quad U_i = \begin{pmatrix} \widetilde{U}_{c_1} R_{c_1} \\ \widetilde{U}_{c_2} R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} \widetilde{V}_{c_1;2} W_{c_1} \\ \widetilde{V}_{c_2;2} W_{c_2} \end{pmatrix}. \tag{5.4}$$

These generators are formed via inter-context communications. See Figure 5.1(d). (5.4) maintains the form of the recursive definition (2.1) of the HSS generators, except that the size has been reduced due to the HSS compression introduced in section 4.

Such a step is then repeated recursively. When the root node is reached, an LU factorization with partial pivoting is performed on $D_i$.

We now examine the communication cost in the HSS factorization. In the first step corresponding to the leaf level, each process performs local QL and LQ factorizations with $U_i$ of size bounded by $m \times r$. No communication is involved. In the subsequent higher levels, the sizes of all the matrices are bounded by $2r \times 2r$, as in Figure 5.1(d) and (5.4). The `ScaLAPACK` QL/LQ factorization and matrix multiplication routines all have the communication cost $[\mathcal{O}(\frac{2r}{b}),\ \mathcal{O}(\frac{(2r)^2}{\sqrt{P_i}})]$ [2], where $b$ is the block size used in `ScaLAPACK`, and $P_i$ is the number of processes used for node $i$ of $\mathcal{T}$. Summing over all the levels, the total cost is bounded by

$$[\mathcal{O}(\frac{r}{b} \log P),\ \mathcal{O}(r^2 \log P)].$$

Since $r \ll n$, this cost is much smaller than that incurred during the HSS compression phase (See (4.10)–(4.11)).

**6. Parallel HSS solution.** We solve the linear system of equations $Ax = b$ after obtaining an HSS approximation to $A$ in Section 4 and the ULV factorization in Section 5. We continue the discussion for the block $2 \times 2$ HSS form in section 5, and the HSS system looks like

$$\begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^H \\ U_{c_2} B_{c_2} V_{c_1}^H & D_{c_2} \end{pmatrix} \begin{pmatrix} x_{c_1} \\ x_{c_2} \end{pmatrix} = \begin{pmatrix} b_{c_1} \\ b_{c_2} \end{pmatrix}. \tag{6.1}$$

With the aid of (5.3), we can rewrite (6.1) into the following form:

$$\begin{pmatrix} \begin{pmatrix} \widetilde{D}_{c_1;1,1} & 0 \\ \widetilde{D}_{c_1;2,1} & \widetilde{D}_{c_1;2,2} \end{pmatrix} & \begin{pmatrix} 0 \\ \widetilde{U}_{c_1} B_{c_1} \begin{pmatrix} \widetilde{V}_{c_2;1}^H & \widetilde{V}_{c_2;2}^H \end{pmatrix} \end{pmatrix} \\ \begin{pmatrix} 0 \\ \widetilde{U}_{c_2} B_{c_2} \begin{pmatrix} \widetilde{V}_{c_1;1}^H & \widetilde{V}_{c_1;2}^H \end{pmatrix} \end{pmatrix} & \begin{pmatrix} \widetilde{D}_{c_2;1,1} & 0 \\ \widetilde{D}_{c_2;2,1} & \widetilde{D}_{c_2;2,2} \end{pmatrix} \end{pmatrix} \begin{pmatrix} \widetilde{x}_{c_1;1} \\ \widetilde{x}_{c_1;2} \\ \widetilde{x}_{c_2;1} \\ \widetilde{x}_{c_2;2} \end{pmatrix} = \begin{pmatrix} \widetilde{b}_{c_1;1} \\ \widetilde{b}_{c_1;2} \\ \widetilde{b}_{c_2;1} \\ \widetilde{b}_{c_2;2} \end{pmatrix}, \tag{6.2}$$

where

$$x_{c_1} = P_{c_1}^H \widetilde{x}_{c_1} = P_{c_1}^H \begin{pmatrix} \widetilde{x}_{c_1;1} \\ \widetilde{x}_{c_1;2} \end{pmatrix}, \quad x_{c_2} = P_{c_2}^H \widetilde{x}_{c_2} = P_{c_2}^H \begin{pmatrix} \widetilde{x}_{c_2;1} \\ \widetilde{x}_{c_2;2} \end{pmatrix},$$
$$b_{c_1} = Q_{c_1} \widetilde{b}_{c_1} = Q_{c_1} \begin{pmatrix} \widetilde{b}_{c_1;1} \\ \widetilde{b}_{c_1;2} \end{pmatrix}, \quad b_{c_2} = Q_{c_2} \widetilde{b}_{c_2} = Q_{c_2} \begin{pmatrix} \widetilde{b}_{c_2;1} \\ \widetilde{b}_{c_2;2} \end{pmatrix}.$$

(6.2) is illustrated by Figure 6.1. We point out that the solution to (6.1) is converted into the solution to (6.2). We can easily compute the original solution $x$ once $\widetilde{x}_{c_1}$ and $\widetilde{x}_{c_2}$ are obtained as follows.

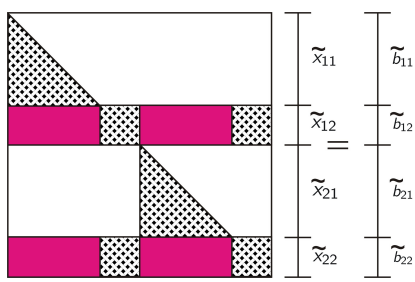



FIG. 6.1. *The illustration of the linear system of equations (6.2) when $i = 3$.*

First, the following two triangular systems can be efficiently solved locally within each context:

$$\widetilde{D}_{c_1;1,1}\ \widetilde{x}_{c_1;1} = \widetilde{b}_{c_1;1}, \quad \widetilde{D}_{c_2;1,1}\ \widetilde{x}_{c_2;1} = \widetilde{b}_{c_2;1}.$$

Then a local update of the right hand side is conducted:

$$\widetilde{b}_{c_1;2} = \widetilde{b}_{c_1;2} - \widetilde{D}_{c_1;2,1}\ \widetilde{x}_{c_1;1}, \quad \widetilde{b}_{c_2;2} = \widetilde{b}_{c_2;2} - \widetilde{D}_{c_2;2,1}\ \widetilde{x}_{c_2;1}.$$

Up to this stage, there is no inter-context communication between $c_1$'s and $c_2$'s contexts.

Next, we update the right hand side via inter-context communication:

$$\widetilde{b}_{c_1;2} = \widetilde{b}_{c_1;2} - \widetilde{U}_{c_1} \left[ B_{c_1} \left( \widetilde{V}_{c_2;1}^H\ \widetilde{x}_{c_2;1} \right) \right],$$
$$\widetilde{b}_{c_2;2} = \widetilde{b}_{c_2;2} - \widetilde{U}_{c_2} \left[ B_{c_2} \left( \widetilde{V}_{c_1;1}^H\ \widetilde{x}_{c_1;1} \right) \right].$$

Finally, we solve a system on the $i$ context:

$$\begin{pmatrix} \widetilde{D}_{c_1;2,2} & \widetilde{U}_{c_1} B_{c_1} \widetilde{V}^H_{c_2;2} \\ \widetilde{U}_{c_2} B_{c_2} \widetilde{V}^H_{c_1;2} & \widetilde{D}_{c_2;2,2} \end{pmatrix} \begin{pmatrix} \widetilde{x}_{c_1;2} \\ \widetilde{x}_{c_2;2} \end{pmatrix} = \begin{pmatrix} \widetilde{b}_{c_1;2} \\ \widetilde{b}_{c_2;2} \end{pmatrix}.$$

This can be done by two triangular solutions after the LU factorization of the coefficient matrix.

For the general case, the above idea is applied recursively.

**7. Performance tests and numerical results.** In this section, we present the parallel performance results of our HSS solver when applied to some dense matrices arising from practical applications. We carry out the experiments on the cluster Cray XE6 (`hopper.nersc.gov`) at the National Energy Research Scientific Computing Center (NERSC). Each node has two 12-core AMD MagnyCours 2.1GHz processors, with 32GB memory. There are 24 cores per node. The timing and storage are reported by IPM (Integrated Performance Monitoring) [19].

EXAMPLE 7.1. *Consider the solution of a Helmholtz equation of the form:*

$$\left(-\Delta - \frac{\omega^2}{v(x)^2}\right) u(x,\omega) = s(x,\omega), \tag{7.1}$$

*where $\Delta$ the Laplacian, $\omega$ the angular frequency, $v(x)$ the seismic velocity field, and $u(x,\omega)$ is called the time-harmonic wavefield solution to the forcing term $s(x,\omega)$.*

Helmholtz equations arise frequently in practical applications such as seismic imaging, where the simplest case of the acoustic wave equation is of the form (7.1). The discretization of the Helmholtz operator often leads to very large sparse matrices $\mathcal{A}$ which are highly indefinite and ill-conditioned. It has been observed that, in the direct factorization of $\mathcal{A}$ the dense intermediate matrix may be compressible [11, 25].

We implemented a parallel multifrontal factorization method for such a matrix $\mathcal{A}$ based on the multifrontal method [9] with the nested dissection reordering [12]. In nested dissection, an $n \times n$ mesh is recursively partitioned into submeshes by separators. The dense matrix $A$ we consider is the last Schur complement corresponding to the final step separator in nested dissection, and also has size $n \times n$. A structured parallel multifrontal method can be further obtained with the intermediate dense matrices (called frontal matrices) approximated by HSS matrices as in [25]. The parallel HSS methods developed in this work can be used to yield a *structured parallel multifrontal method.*

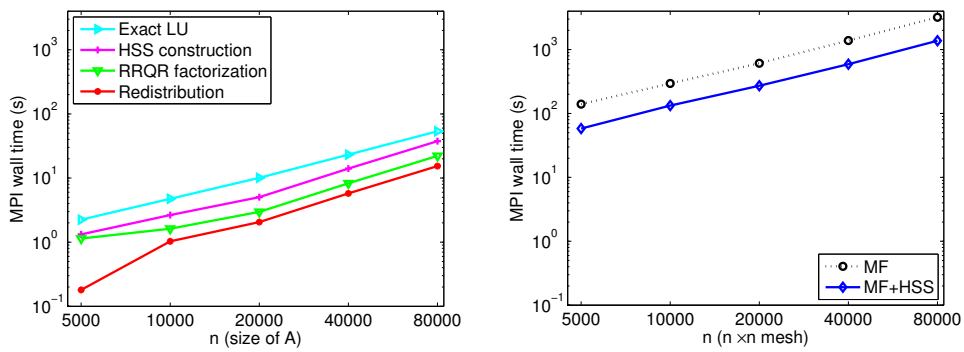
First, we present the performance of the HSS algorithms applied to the dense matrix $A$. (The frequency $\omega = 5Hz$ is used to get $\mathcal{A}$ and then $A$.) For the weak scaling test, we increase the number of cores by a factor of four upon doubling the mesh size $n$. Table 7.1 shows the runtime of the parallel HSS methods for $n$ ranging from $5,000$ to $80,000$. We split the total HSS compression time into an RRQR factorization part and a data redistribution part. As predicted, the HSS factorization and solution are faster than the HSS compression. Inside the compression phase, the redistribution part takes less time than the RRQR factorization part. The weak scaling is depicted in Figure 7.1(a). We observe that the algorithms scale about the same. The weak scaling factor is about 2.0.

Secondly, we study the performance gain of the structured parallel multifrontal solver for the sparse discretized matrix $\mathcal{A}$ (with parallel HSS operations for the intermediate dense matrices). We consider mesh sizes in 2D ranging from $5,000 \times 5,000$ to

| $n$ | | | $5,000$ | $10,000$ | $20,000$ | $40,000$ | $80,000$ |
|---|---|---|---|---|---|---|---|
| $P$ | | | 16 | 64 | 256 | $1,024$ | $4,096$ |
| Exact LU factorization (s) | | | 2.24 | 4.75 | 10.09 | 23.18 | 53.65 |
| HSS | Construction | Total (s) | 1.32 | 2.65 | 5.03 | 14.03 | 37.52 |
| | | RRQR (s) | 1.14 | 1.62 | 2.97 | 8.28 | 22.14 |
| | | Redistribution (s) | 0.18 | 1.03 | 2.06 | 5.75 | 15.38 |
| | Factorization (s) | | 0.11 | 0.14 | 0.19 | 0.24 | 0.29 |
| | Solution (s) | | 0.07 | 0.10 | 0.14 | 0.20 | 0.23 |

TABLE 7.1

*Computational time and weak scaling of the parallel HSS algorithms applied to the largest dense frontal matrices $A$ from the exact multifrontal factorization of 2D discretized Helmholtz operators in (7.1), where $n$ is the size of $A$, $P$ is the number of processes, and the relative tolerance in HSS methods is $\tau = 10^{-4}$.*




(a) HSS results for $A$ as in Tables 7.1 (b) MF+HSS results for $\mathcal{A}$ as in Table 7.2

FIG. 7.1. *Weak scaling curves for the runtime data in Tables 7.1 and 7.2.*

$80,000 \times 80,000$. (The largest matrix $\mathcal{A}$ has dimension 6.4 billion.) Table 7.2 shows both the runtime and the memory usage for the solver, as compared with the exact multifrontal solver. It is observed that the structured solver is about 2 to 3 times faster than the exact multifrontal solver, and requires less storage.

| $N$ (mesh: $n \times n$) | | $5,000$ | $10,000$ | $20,000$ | $40,000$ | $80,000$ |
|---|---|---|---|---|---|---|
| $P$ | | 16 | 64 | 256 | $1,024$ | $4,096$ |
| MF | Time (s) | $1.40e2$ | $2.95e2$ | $6.12e2$ | $1.39e3$ | $3.20e3$ |
| | Memory (GB) | $4.20e1$ | $1.87e2$ | $8.06e2$ | $3.43e3$ | $1.42e4$ |
| MF+HSS | Time (s) | $5.80e1$ | $1.33e2$ | $2.72e2$ | $5.91e2$ | $1.38e3$ |
| | Memory (GB) | $3.20e1$ | $1.40e2$ | $5.95e2$ | $2.46e3$ | $9.89e3$ |

TABLE 7.2

*Parallel computational time and storage of the structured multifrontal method (*MF+HSS*) for the 2D discretized Helmholtz matrix $\mathcal{A}$ as compared with the exact multifrontal method (* MF*), where $\mathcal{A}$ has size $n^2$ and $P$ is the number of processes.*

Thirdly, we also solve the Helmholtz equation discretized on a 3D $n \times n \times n$ mesh, for $n$ ranging from 100 to 500. The result is reported in Table 7.3.

EXAMPLE 7.2. *We show the accuracy of the structured parallel multifrontal solver*

| $n$ (mesh: $n \times n \times n$) | | 100 | 200 | 300 | 400 | 500 |
|---|---|---|---|---|---|---|
| $P$ | | 64 | 256 | 1,024 | 4,096 | 8,192 |
| MF | Time (s) | 1.02$e$2 | 1.71$e$3 | 5.11$e$3 | Out of memory | |
| | Memory (GB) | 4.00$e$1 | 5.64$e$2 | 2.83$e$3 | | |
| MF+HSS | Time (s) | 1.00$e$1 | 1.06$e$3 | 3.22$e$3 | 6.41$e$3 | 9.83$e$3 |
| | Memory (GB) | 3.70$e$1 | 5.10$e$2 | 2.40$e$3 | 6.17$e$3 | 1.61$e$4 |

TABLE 7.3

*Parallel computational time and storage of the structured multifrontal factorization (*MF+HSS*) for the 3D discretized Helmholtz matrix $\mathcal{A}$ as compared with the exact multifrontal method (*MF*), where $\mathcal{A}$ has size $n^3$, $P$ is the number of processes, and* MF *runs out of memory for $n \geq 400$ with the given number of processes.*

*(with HSS methods) for the time-harmonic waves in 2D in the previous example. The mesh size is* $5,000 \times 3,000$.

Figure 7.2(a) displays a 5Hz time-harmonic wavefield solution to the 2D Helmholtz equations using the structured multifrontal solver, with the relative tolerance $\tau = 10^{-2}$. The amplitude difference between the solution in Figure 7.2(a) and the true solution is displayed in Figure 7.2(b). We note that 2 digits of accuracy is insufficient to produce an acceptable wavefield solution. In Figure 7.2(c), we display the solution with $\tau = 10^{-4}$. The amplitude difference is shown in Figure 7.2(d) and is a satisfactory result that is generally sufficient for seismic applications.

EXAMPLE 7.3. *We present the strong scaling result of the parallel HSS construction for a fixed dense Toeplitz matrix*

$$A = (a_{i-j})_{n \times n}\,.$$

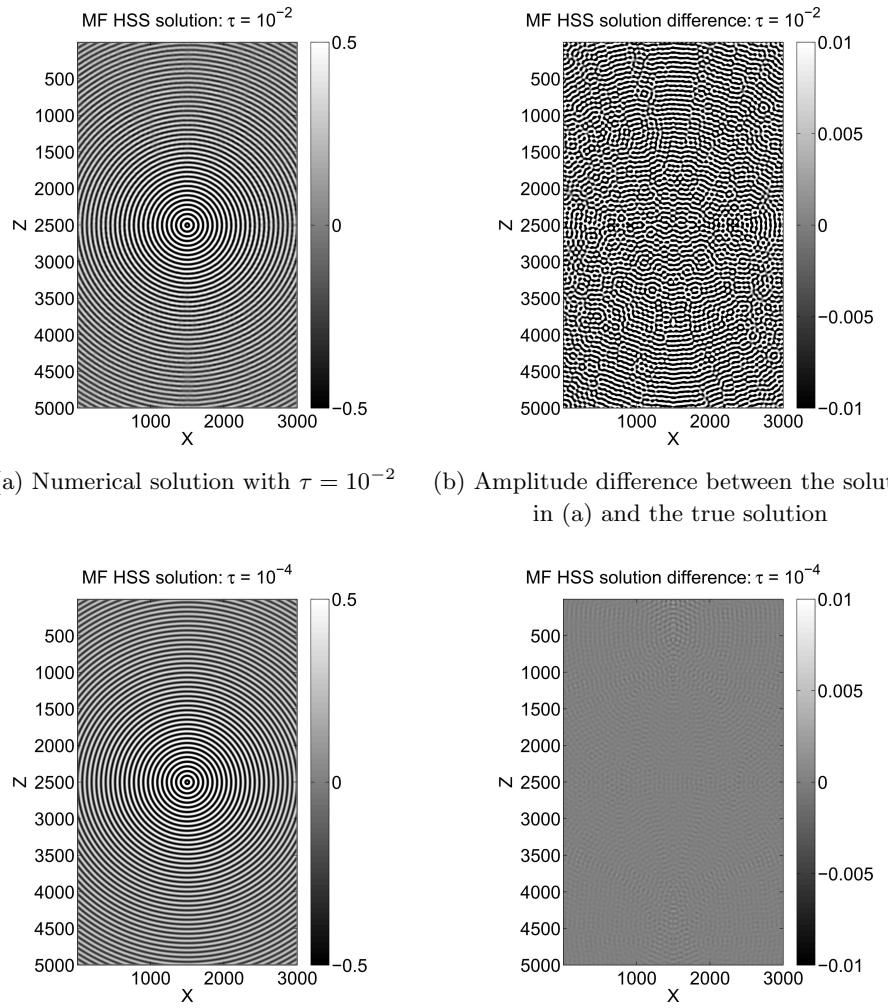
Studies of the low-rank property related to Toeplitz matrices can be found in [21, 8, etc.]. We test the HSS construction only. For $n = 100,000$, the computation time is given in Table 7.4, and the strong scaling curve is plotted in Figure 7.3. We note that the data redistribution part is more efficient than the RRQR factorization part in the HSS construction.

| $P$ | 64 | 128 | 256 | 512 | 1024 |
|---|---|---|---|---|---|
| HSS construction (s) | 90 | 51 | 32 | 25 | 28 |
| RRQR factorization (s) | 62 | 35 | 22 | 18 | 15 |
| Redistribution (s) | 28 | 16 | 10 | 7 | 13 |

TABLE 7.4

*Parallel HSS construction time and strong scaling for a* $100,000 \times 100,000$ *Toeplitz matrix.*

**8. Conclusions.** We have designed and implemented novel parallel algorithms for the HSS structured matrix algorithms in parallel computation. We are able to conduct classical structured compression, factorizations, and solutions in parallel. We performed detailed analysis of the communication costs of the parallel algorithms, and found that the algorithms are more communication bound than the other algorithms without using the rank structures. This is mainly because the amount of floating-point operation is drastically reduced by exploiting the low rankness, but the reduction in communication is moderate. The future challenge is to design novel communication-avoiding algorithms for this class of rank structured methods.




(a) Numerical solution with $\tau = 10^{-2}$

(b) Amplitude difference between the solution in (a) and the true solution

(c) Numerical solution with $\tau = 10^{-4}$

(d) Amplitude difference between the solution in (c) and the true solution

FIG. 7.2. *5Hz time-harmonic wavefield solutions and the corresponding errors for the 2D Helmholtz equation using the structured parallel multifrontal solver with different tolerances $\tau$, where the results are displayed as images with the Matlab function* `imagesc`.

Our implementations are portable by using the two well established libraries, `BLACS` and `ScaLAPACK`. The computational results for weak scaling, strong scaling, and accuracy demonstrate the high efficiency and scalability of our algorithms. The algorithms are very useful in solving large dense and sparse linear systems that arise in real-world applications. For example, we have appplied our new parallel HSS-embedded multifrontal solver to the anisotropic Helmholtz equations for seismic imaging, and were able to solve a linear system with 6.4 billion unknowns using 4096 processors, in about 20 minutes. The classical multifrontal solver simply failed due to high demand of memory. Our techniques can also benefit the development of fast parallel methods using the other rank structures.

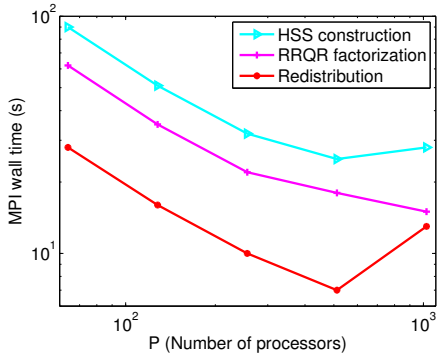



FIG. 7.3. *Strong scaling curve for the HSS construction for a dense* $100,000 \times 100,000$ *Toeplitz matrix. The data is from table 7.4.*

**9. Acknowledgements.** We thank the members of the Geo-Mathematical Imaging Group (GMIG) at Purdue University, ConocoPhillips, ExxonMobil, PGS, Statoil and Total, for the partial financial support. This research used resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Science of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. The research of Jianlin Xia was supported in part by NSF grants DMS-1115572 and CHE-0957024.

REFERENCES

[1] T. Bella, Y. Eidelman, and V. Gohberg, I. and. Olshevsky, *Computations with quasiseparable polynomials and matrices*, Theoret. Comput. Sci., 409 (2008), pp. 158–179.

[2] L. S. Blackford, J. Choi, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*, SIAM, Philadelphia, 1997. 325 pages.

[3] *http://www.netlib.org/blacs.*

[4] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem, 27 (2003), pp. 405–422.

[5] S. Börm and W. Hackbusch, *Data-sparse approximation by adaptive* $\mathcal{H}^2$*-matrices*, Technical report, Leipzig, Germany: Max Planck Institute for Mathematics, 86 (2001).

[6] S. Chandrasekaran, P. Dewilde, M. Gu, T. Pals, X. Sun, A.-J. van der Veen, and D. White, *Some fast algorithms for sequentially semiseparable representations*, SIAM J. Matrix Anal. Appl., 27 (2005), pp. 341–364.

[7] S. Chandrasekaran, M. Gu, and T. Pals, *A fast ulv decomposition solver for hierarchically semiseparable representations*, SIAM J. Matrix Anal. Appl., 28 (2006), pp. 603–622.

[8] S. Chandrasekaran, M. Gu, X. Sun, J. Xia, and J. Zhu, *A superfast algorithm for toeplitz systems of linear equations*, SIAM J. Matrix Anal. Appl., (2007), pp. 1247–1266.

[9] I. S. Duff and J. K. Reid, *The multifrontal solution of indefinite sparse symmetric linear equations*, ACM Trans. Math. Software, 9 (1983), pp. 302–325.

[10] Y. Eidelman and I. Gohberg, *On a new class of structured matrices*, Integr. Equat. Operat. Theor., 34 (1999), pp. 293–324.

[11] B. Engquist and L. Ying, *Sweeping preconditioner for the helmholtz equation: hierarchical matrix representation*, to appear, Commun. Pure Appl. Math., 64 (2011), pp. 697–735.

[12] J. A. George, *Nested dissection of a regular finite element mesh*, SIAM J. Numer. Anal., 10 (1973), pp. 345–363.

[13] G. H. Golub and C. V. Loan, *Matrix computations*, The Johns Hopkins University Press, Baltimore, MD, 3rd edition, (1996).

[14] M. Gu and S. C. Eisenstat, *Efficient algorithms for computing a strong rank-revealing qr factorization*, SIAM J. Sci. Comput., 17 (1996), pp. 848–869.

[15] W. Hackbusch, *A sparse matrix arithmetic based on* $\mathcal{H}$*-matrices. Part I: introduction to*

*$\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.

[16] W. Hackbusch, L. Grasedyck, and S. Börm, *An introduction to hierarchical matrices*, Math. Bohem., 127 (2002), pp. 229–241.

[17] W. Hackbusch, B. Khoromskij, and S. A. Sauter, *On $\mathcal{H}^2$-matrices*, Lectures on applied mathematics (Munich, 1999), Springer, Berlin, (2000), pp. 9–29.

[18] W. Hackbusch and B. N. Khoromskij, *A sparse $\mathcal{H}$-matrix arithmetic. Part-II: Application to multi-dimensional problems*, Computing, 64 (2000), pp. 21–47.

[19] *http://ipm-hpc.sourceforge.net/*.

[20] *http://www.netlib.org/lapack*.

[21] P. G. Martinsson, V. Rokhlin, and M. Tygert, *A fast algorithm for the inversion of general toeplitz matrices*, Comput. Math. Appl., 50 (2005), pp. 741–752.

[22] *http://www.mpi-forum.org*.

[23] *http://www.netlib.org/scalapack*.

[24] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi, *A bibliography on semiseparable matrices*, Calcolo, 42 (2005), pp. 249–270.

[25] S. Wang, M. V. De Hoop, and J. Xia, *Seismic inverse scattering via helmholtz operator factorization and optimization*, J. Computat. Phys., 229 (2010), pp. 8445–8462.

[26] S. Wang, M. V. de Hoop, and J. Xia, *On 3d modeling of seismic wave propagation via a structured massively parallel multifrontal direct helmholtz solver*, Geophys. Prospect., 59 (2011), pp. 857–873.

[27] J. Xia, *On the complexity of some hierarchical structured matrices*, http://www.math.purdue.edu/∼xiaj/work/hsscost.pdf, submitted to SIAM J. Matrix Anal. Appl., (2011).

[28] ———, *A robust inner-outer hss preconditioner*, Submitted to Numer. Linear Algebra Appl., (2011).

[29] J. Xia, S. Chandrasekaran, M. Gu, and X. S. Li, *Superfast multifrontal method for large structured linear systems of equations*, SIAM J. Matrix Anal. Appl., 31 (2009), pp. 1382–1411.

[30] ———, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 2010 (2010), pp. 953–976.

[31] J. Xia and M. Gu, *Robust approximate cholesky factorization of rank-structured symmetric positive definite matrices*, SIAM J. Matrix Anal. Appl., 31 (2010), pp. 2899–2920.

## DISCLAIMER

This document was prepared as an account of work sponsored by the United States Government. While this document is believed to contain correct information, neither the United States Government nor any agency thereof, nor the Regents of the University of California, nor any of their employees, makes any warranty, express or implied, or assumes any legal responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by its trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or any agency thereof, or the Regents of the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or any agency thereof or the Regents of the University of California.