Proceedings of the Project Review, Geo-Mathematical Imaging Group (Purdue University, West Lafayette IN), Vol. 1 (2011) pp. 159-174.

EFFICIENT PARALLEL ALGORITHMS FOR HIERARCHICALLY SEMISEPARABLE MATRICES

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

Abstract. Recently, hierarchically semiseparable (HSS) matrices have been used in the development of fast direct sparse solvers. Key applications of HSS algorithms, coupled with multifrontal solvers, appear in solving certain large-scale computational inverse problems. Here, we develop massively parallel HSS algorithms appearing in these solution methods, namely, parallel HSS construction using the rank revealing QR (RRQR) method, parallel HSS ULV factorization, and parallel HSS solution. HSS representations have a nice binary tree structure called HSS tree. HSS operations can be conducted following the traversal of this tree, and communications are generally limited to between siblings and between parents and children. Thus, HSS algorithms are often highly scalable. `BLACS` [1] and `ScaLapack` [3] are used as our porTable libraries. We construct *contexts* (sub-communicators) on each node of the HSS tree and exploit the governing 2D-block-cyclic data distribution scheme widely used in `ScaLapack`. Computational examples confirm the weak scaling, strong scaling and accuracy of our implementation.

Key words. parallel algorithm, HSS matrix, low rank, compression, 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 developing fast solutions of partial differential equations, integral equations, and companion eigenvalue problems. Several useful rank structured matrix representations have been developed, including $\mathcal{H}$-matrices [11, 16, 12], $\mathcal{H}^2$-matrices [5, 6, 15], quasiseparable matrices [4, 10], and semiseparable matrices [7, 14].

Here, we focus on the hierarchically semiseparable (HSS) matrices, in the context of fast sparse direct solvers. Key applications of HSS algorithms, coupled with massively parallel multifrontal solvers, appear in solving certain large-scale computational inverse problems. We mention the multi-frequency formulation of (seismic) inverse scattering and tomography. Here, the 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 accuracy of computation can be limited, namely, in concert with the accuracy of the data. The frequency formulation is attractive, in as much as that frequency essentially appears in the convergence criterion for the iteration; hence, one can exploit a relationship between data and components (scale of variation) of the coefficients that can be reconstructed.

HSS representations have a tree structure, called the HSS tree, and HSS operations can be generally conducted following the traversal of this tree. The traversal can be subjected to parallelization of the corresponding algorithms. Existing studies of HSS structures mostly focus on the mathematical aspects of HSS methods; here, we present new efficient, parallel HSS algorithms and analyze computational aspects in particular scalability.

We concentrate on three, connected, algorithms: The parallel construction of an HSS form from a general dense matrix, the parallel ULV factorization of such a matrix, and the parallel solution to multiple right-hand sides. Since the HSS algorithms mostly consist of dense matrix kernels, we choose `BLACS` [1] and `ScaLapack` [3] as our porTable libraries. We construct *contexts* (sub-communicators) on each node of the HSS tree. We also exploit the governing 2D-block-cyclic data distribution scheme widely used in `ScaLapack` to achieve high performance. In [19] it is proven

*Center for Computational and Applied Mathematics, Purdue University, 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. D-AC02-05CH11231.

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

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

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

that the complexity of HSS construction, factorization and solution 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.

The outline of the paper is as follows. In Section 2, we present an overview of HSS structures. The parallelization strategy and the performance model are introduced in Section 3, where we also briefly discuss the portability of `BLACS` and `ScaLapack` which enables us to achieve high performance. In Section 4, we present our parallel HSS construction framework which consists of three phases: parallel rank revealing QR (RRQR) factorization using the modified Gram-Schmidt (MGS) method, the parallel row construction and the parallel column construction. The parallel HSS factorization is presented in Section 5, in which we discuss a generalization involving the use of two children's contexts c_1, c_2 and the parent context i. The communication patterns are composed of intra-context and inter-context ones. In Section 6, we describe the parallel solution strategy. We present computational experiments in Section 7 and confirm the weak scaling of large Helmholtz problems, the accuracy and the strong scaling of a large dense Toeplitz matrix. The results show that our code achieve high performance when the matrix is scaled to up to 6.4 billions.

2. Overview of HSS structures. We follow the work of [19, 17] and briefly summarize the key concepts in HSS structures. Let A be a general $n \times n$ real or complex matrix and $\mathcal{I} = \{1, 2, ..., n\}$ be the set of all indices. Suppose $t_i \subset \mathcal{I}$ and $t_j \subset \mathcal{I}$ are subsets of $\mathcal{I}$, we denote the submatrix of A with row index subset t_i and column index subset t_j as $A|_{t_i \times t_j}$. Suppose $\mathcal{T}$ is a full binary tree with k leaf nodes, then the number of nodes on $\mathcal{T}$ is $2k-1$. We say that $\mathcal{T}$ is in its postordering form if for each non-leaf node i among $\{1, 2, ..., 2k-1\}$, its left child c_1 and right child c_2 satisfy $c_1 < c_2 < i$. Moreover, the node numbers within each subtree are consecutive. Then we have the following definition [19]:

DEFINITION 2.1. *We define $\mathcal{T}$ as an HSS tree and A is represented in HSS form if the following conditions are satisfied:*

- *$\mathcal{T}$ is in its postordering form.*
- *Suppose a non-leaf node i represents the submatrix $A|_{t_i \times t_i}$, and child c_1 represents $A|_{t_{c_1} \times t_{c_1}}$, child c_2 represents $A|_{t_{c_2} \times t_{c_2}}$, then $t_{c_1} \cap t_{c_2} = \emptyset, t_{c_1} \cup t_{c_2} = t_i$.*
- *The root node $2k-1$ represents the entire matrix A, $t_{2k-1} = \{1, 2, ..., n\}$.*
- *There exist matrices $D_i, U_i, R_i, B_i, W_i, V_i$ (called HSS generators) associated with each node i on $\mathcal{T}$, such that:*

$$D_i = A|_{t_i \times t_i} \approx \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^H \\ U_{c_2} B_{c_2} V_{c_1}^H & D_{c_2} \end{pmatrix}, \qquad D_{2k-1} = A, \tag{2.1}$$

$$U_i = \begin{pmatrix} U_{c_1} R_{c_1} \\ U_{c_2} R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} V_{c_1} W_{c_1} \\ V_{c_2} W_{c_2} \end{pmatrix},$$

where upper script H denotes the Hermitian transpose.

For example, Figure 1(a) illustrates a block 2×2 HSS representation of A where the number of leaf nodes in the HSS tree $\mathcal{T}$ is $k = 2$:

$$A \approx \begin{pmatrix} D_1 & U_1 B_1 V_2^H \\ U_2 B_2 V_1^H & D_2 \end{pmatrix}.$$

Figure 1(b) illustrates a block 4×4 HSS representation of A where the number of leaf nodes in the HSS tree $\mathcal{T}$ is $k = 4$:

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

For some applications like the multifrontal method [9, 13, 18], we are interested in obtaining the Schur complement of A. For instance, the following block 3×3 HSS representation of A where

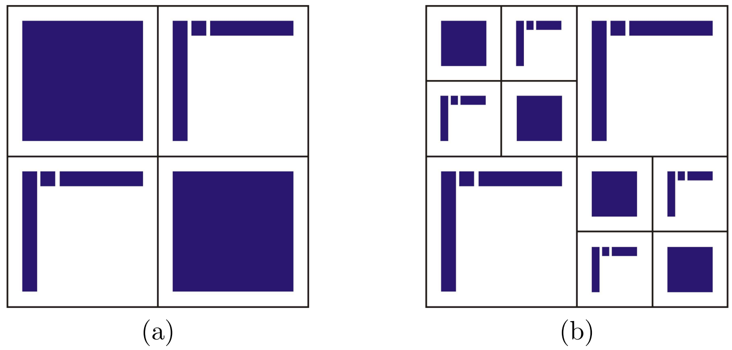

FIG. 1. *(a). a block 2×2 HSS representation of A; (b). a block 4×4 HSS representation of A.*

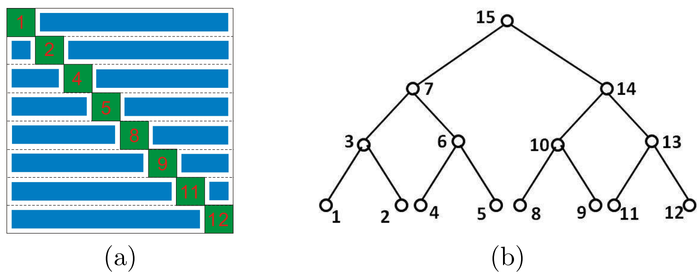

FIG. 2. *(a). a block 8×8 representation of a dense matrix A; (b). the illustration of the HSS tree $\mathcal{T}$ associated with A for a block 8×8 case.*

the number of leaf nodes in the HSS tree $\mathcal{T}$ is $k = 3$ treats D_4 as the Schur complement of A:

$$A \approx \begin{pmatrix} \begin{pmatrix} D_1 & U_1 B_1 V_2^H \\ U_2 B_2 V_1^H & D_2 \end{pmatrix} & \begin{pmatrix} U_1 R_1 \\ U_2 R_2 \end{pmatrix} B_3 V_4^H \\ U_4 B_4 \begin{pmatrix} W_1^H V_1^H & W_2^H V_2^H \end{pmatrix} & D_4 \end{pmatrix}.$$

3. Parallelization strategy. Let n be the matrix dimension, m is the row block size corresponding to the leaf nodes of the HSS tree. We choose m first, which is related to the HSS rank, then choose the number of processes $P \approx n/m$. In this paper, we assume that P is a power of two, and the logorithm is in base 2. The parallel operations can be organized effectively using the HSS tree $\mathcal{T}$, via either upward sweeping or downward sweeping. We refer to the bottom level of the tree as level 1, and the next level up as level 2, and so on. We use an example in Fig. 2 to illustrate the organization of the algorithms. The matrix is partitioned into eight block rows (Fig. 2(a)), with its HSS tree $\mathcal{T}$ displayed in Fig. 2(b). We use eight processes $\{0, 1, 2, 3, 4, 5, 6, 7\}$ for the parallel operations. Each process P_i works on the leaf node i 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 the level-3 nodes, and so on.

3.1. Using `ScaLAPACK` and `BLACS` . Since the HSS algorithms mostly consist of dense matrix kernels, we chose to use as much as as possible the well established routines in the `ScaLAPACK` library [3] and its communication substrate, the `BLACS` library [1]. The governing distribution scheme is 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 the `BLACS` term, similar to the sub-communicator concept in the MPI standard. 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 Fig. 2, the eight processes can be arranged as eight contexts for the leaf nodes in $\mathcal{T}$. Four contexts are defined at the second level: {0, 1} $\leftrightarrow$ node 3, {2, 3} $\leftrightarrow$ node 6, {4, 5} $\leftrightarrow$ node 10, and {6, 7} $\leftrightarrow$ node 13. Two contexts are defined at the third level: {0, 1 ; 2, 3} $\leftrightarrow$ node 7, and {4, 5 ; 6, 7} $\leftrightarrow$ node 14. Finally, one context is defined for node 15: [0,1,4,5 ; 2,3,6,7]. Here the notation {0, 1 ; 2, 3} means processes 1 and 2 are stacked atop processes 2 and 3. 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 *resistribution* to merge the data distributed in the two children's process contexts to the parent's context. Since the two children's contexts have the same size and shape and the parent context doubles each child's context, the parent context can be arranged to combine the two children's contexts either side by side or one atop the other. Thus, the processes grid is mantained as square as possible, and the redistribution pattern is simple, which only involves *pairwise exchange*. That is, a pair of processes at the same coordinate in the two children 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 our parallel algorithms. r is the HSS rank, i.e., the maximum off-diagonal rank after compression. The communication cost is modeled as follows.

- We use the pair [`#messages`, `#words`] to count the number of messages and the number of words transferred in a parallel algorithm. The parallel runtime can be modeled as the following (ignoring the overlap of communication with computation):

$$Time = \texttt{\#flops} \cdot \gamma + \texttt{\#messages} \cdot \alpha + \texttt{\#words} \cdot \beta \ , \tag{3.1}$$

  where, γ is the time taken for each flop, α is the time taken for each message (latency), and β is the time taken for each word transferred (reciprocal bandwidth).
- 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 how to construct the HSS representation of A introduced in [18, 19] from the bottom up in parallel. The construction is composed of the row compression step (Section 4.2) followed by the column compression step (Section 4.3). The key kernel is the rank revealing QR algorithm which we discuss first in Section 4.1.

4.1. Parallel rank revealing QR (RRQR). The key step to obtain the HSS representation of A is to compress the off-diagonal block of D_i, denoted as $F_i = A|_{t_i \times (\mathcal{I} \setminus t_i)}$. Truncated SVD is one option to realize such a compression. That is, we drop those singular values below a prescribed threshold after the full SVD factorization of F_i is computed. This is 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.

Consider the ith off-diagonal block $F_i = (f_1, f_2, ..., f_J)$ associated with node i in the HSS tree $\mathcal{T}$. Let r_i be the numerical rank determined by the tolerance. Assume F_i is of size $M \times N$, and is distributed on the process context $P \approx \sqrt{P} \times \sqrt{P}$. That is, the local dimension of F_i is $\frac{M}{\sqrt{P}} \times \frac{N}{\sqrt{P}}$. The following algorithm, based on Modified Gram-Schmidt [8], computes RRQR in parallel: $F_i \approx \widetilde{Q}\widetilde{R}$ where $\widetilde{Q} = (q_1, q_2, ..., q_{r_i}), \widetilde{R}^H = (r_1^H, r_2^H, ..., r_{r_i}^H)$.

```
for j = 1:r_i
    1.  In parallel, search for the column f_j with the maximum norm;
    2.  normalize f_j to q_j:   q_j = f_j/||f_j||,  r_jj = ||f_j||;
    3.  broadcast q_j within the context associated with the node i;
    4.  PBLAS2:  r_j = q_j^H F_i;
    5.  rank one update:   F_i = F_i - q_j r_j .
end
```

Communications occur in Steps 1 and 3. The other steps involve local computations only. In Step 1, the processes in each column group perform $\frac{N}{\sqrt{P}}$ reductions to compute the column norms, with communication cost $= [\log\sqrt{P},\ \log\sqrt{P}]\cdot\frac{N}{\sqrt{P}}$. This is followed by another reduction among the processes in the row group, 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 $[\log\sqrt{P},\ \frac{M}{\sqrt{P}}\log\sqrt{P}]$.

Adding the leading terms, we obtain the following communication cost:

$$RRQR_{comm} = \left[\frac{N}{\sqrt{P}}\log\sqrt{P},\ \frac{M+N}{\sqrt{P}}\log\sqrt{P}\right] . \tag{4.1}$$

To achieve higher performance, a block RRQR strategy can be adopted similarly, like `Lapack` [2] subroutine `xGEQP3`.

4.2. Parallel row compression. We still use the 8×8 block matrix in Fig. 2 to illustrate the algorithm step by step.

Row compression—Step 1. In the first step, all the leaf nodes 1, 2, 4, 5, 8, 9, 11, 12 have their own process contexts: $\{0\},\{1\},\{2\},\{3\},\{4\},\{5\},\{6\}$ and $\{7\}$. Each process owns part of the global matrix A, given by $D_i = A|_{t_i\times t_i}, F_i = A|_{t_i\times(\mathcal{I}\backslash t_i)}$. F_i s are indicated by the shaded areas in Figure 2(a). We perform RRQR (Section 4.1) on F_i:

$$F_i \approx U_i\widehat{F}_i\ , \tag{4.2}$$

where $\widehat{F}_i$ can also be denoted as $A|_{\hat{t}_i\times(\mathcal{I}\backslash t_i)}$ and $\hat{t}_i$ is the update of t_i due to the rank contraction. This first step is done locally within each process. One of the *HSS generators* U_i is obtained here.

Moving to the second level of $\mathcal{T}$, the compressions must be carried out among a pair of processes in each context. To prepare for this, we need a *redistribution* phase prior to the compressions. In redistribution, 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 $3,6,10,13$ whose contexts are $\{0,1\}$, $\{2,3\}$, $\{4,5\}$ and $\{6,7\}$ respectively. The level-2 off-diagonal blocks $F_i, i=3,6,10,13$ are formed by merging $\widehat{F}_{c_1}$ and $\widehat{F}_{c_2}$ via the following formula:

$$F_i = \begin{pmatrix} A|_{\hat{t}_{c_1}\times(\mathcal{I}\backslash t_i)} \\ A|_{\hat{t}_{c_2}\times(\mathcal{I}\backslash t_i)} \end{pmatrix}, \qquad F_{c_1} = A|_{\hat{t}_{c_2}\times t_{c_1}}, \qquad F_{c_2} = A|_{\hat{t}_{c_1}\times t_{c_2}}. \tag{4.3}$$

This procedure is illustrated in Figure 3(a). Two communication steps are needed. The first step is to generate F_{c_1} and F_{c_2} by exchanging $A|_{\hat{t}_{c_1}\times t_{c_2}}$ 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. The second step is to redistribute the newly merged off-diagonal block F_i onto the process grid associated with node i's context, for $i=3,6,10$ or 13. 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 $\frac{r\,n}{2}\cdot 2$. The communication cost is $[2,\ \frac{r\,n}{2}\cdot 2]$.

Row compression—Step 2. At level 2 of the tree, the contexts $\{0,1\}$, $\{2,3\}$, $\{4,5\}$ and $\{6,7\}$ are associated with the nodes $3,6,10,13$, respectively. The distribution of the off-diagonal

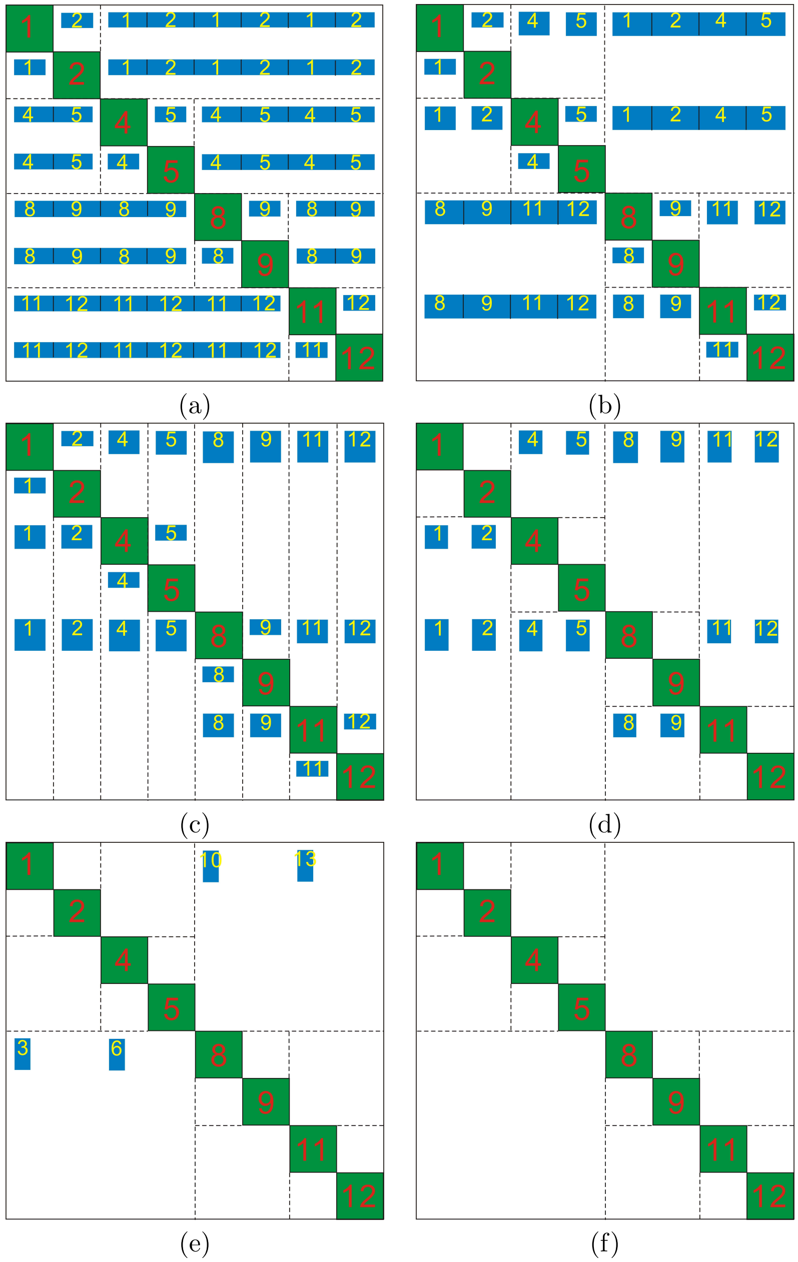

FIG. 3. *(a). parallel row construction 1; (b). parallel row construction 2; (c). parallel row construction 3 and initiate the parallel column construction; (d). parallel column construction 1; (e). parallel column construction 2; (f). parallel column construction 3 and finalize the entire parallel HSS construction.*

blocks $F_i, i = 3, 6, 10, 13$ is finished, as shown in Fig. 3(a). We then perform the parallel RRQR within each context for each F_i:

$$F_i \approx \begin{pmatrix} R_{c_1} \\ R_{c_2} \end{pmatrix} \widehat{F}_i \,, \tag{4.4}$$

where $\widehat{F}_i$ can also be denoted as $A|_{\hat{t}_i \times (\mathcal{I} \setminus t_i)}$, $\hat{t}_i$ is the row index subset of $\widehat{F}_i$. One of the *HSS* *generators* R_i is obtained here. Since each F_i is bounded by size $2r \times n$ and two processes are used for each RRQR, using Eqn.(4.1), we obtain the communication cost to be $[\frac{n}{\sqrt{2}} \log \sqrt{2},\ \frac{2r+n}{\sqrt{2}} \log \sqrt{2}]$.

To prepare for the third level HSS construction, 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, $0 \leftrightarrow 1$ and $2 \leftrightarrow 3$ exchanges occur simultaneously. There is no need for data exchanges between processes 0 and 3, or processes 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. This procedure is illustrated in Fig. 3(b). There are also two communication steps similar to Eqn.(4.3).

During the redistribution phase, the number of messages is 2, and the number of words exchanged is $\frac{r\,n}{4} \cdot 2$. The communication cost is $[2,\ \frac{r\,n}{4} \cdot 2]$.

Row compression—Step 3. At level 3, there are only two contexts $\{0,1;2,3\}$ and $\{4,5;6,7\}$ which are associated with the nodes 7 and 14 respectively. Each of the two off-diagonal blocks $F_i, i = 7, 14$ has already been distributed onto the respective process context (see Fig. 3(b)). We then perform the parallel RRQR within each context for each F_i, similar to Eqn.(4.4). One of the *HSS generators* R_i is also obtained here. The communication cost of RRQR is given by $[\frac{n}{\sqrt{4}} \log \sqrt{4},\ \frac{2r+n}{\sqrt{4}} \log \sqrt{4}]$.

Since the upper level node is the root node 15 of $\mathcal{T}$, there is no off-diagonal block F_{15} 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 eq.(4.3) except that there is no merging step to form F_{15}. The procedure is illustrated in Fig. 3(c). The communication cost in the redistribution phase is $[2,\ \frac{r\,n}{8} \cdot 2]$.

To summarize, we now sum all the messages and number of words communicated at all the levels of the tree. Denote $L = \log P$ as the number of levels in $\mathcal{T}$. Then, the total communication cost is summed up by the following.

1. Redistribution

$$\texttt{\#messages} = \sum_{i=1}^{L} 2 = 2 \log P \tag{4.5}$$

$$\texttt{\#words} = \sum_{i=1}^{L} (\frac{r\,N}{2^i} 2) = \mathcal{O}(2r\ N) \tag{4.6}$$

2. RRQR

$$\texttt{\#messages} = \sum_{i=1}^{L} \frac{n}{\sqrt{2^i}} \log \sqrt{2^i} = \frac{n}{2} \sum_{i=1}^{L} \frac{i}{2^{i/2}} = \mathcal{O}(\frac{n}{2} \log P) \tag{4.7}$$

$$\texttt{\#words} = \sum_{i=1}^{L} \frac{2r+n}{\sqrt{2^i}} \log \sqrt{2^i} = \frac{2r+n}{2} \sum_{i=1}^{L} \frac{i}{2^{i/2}} = \mathcal{O}(\frac{2r+n}{2} \log P) \tag{4.8}$$

4.3. Parallel column compression. After the row compression, the matrices remained to be compressed are much smaller, and the communication cost is also lower in this step. We now describe how the column compression works using the same 8×8 block matrix example.

Column compression—Step 1.

After the parallel row construction, the updated off-diagonal blocks $F_i, i = 1, 2, ...14$ are stored in the individual contexts. 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 $\{0,1;2,3\}$. Since the algorithms for both row and column compressions are upward sweeping along $\mathcal{T}$ [19], there is a redistribution step for the leaf off-diagonal blocks from their parents' contexts to their own contexts. For instance, consider the context $\{0\}$ associated with the leaf node 1, the redistribution procedure can be formulated as:

$$F_1 = \begin{pmatrix} A|_{\hat{t}_2 \times t_1} \\ A|_{\hat{t}_6 \times t_1} \\ A|_{\hat{t}_{14} \times t_1} \end{pmatrix}, \qquad \hat{t}_1 = \hat{t}_2 \cup \hat{t}_6 \cup \hat{t}_{14}. \tag{4.9}$$

Eq.(4.9) can be similarly applied to other contexts associated with the leaf nodes. We still rely on the subroutine `PxGEMR2D` to realize this inter-contexts communications.

After the redistribution, the layout of the off-diagonal blocks is illustrated by Figure 3(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 $i, i = 1, 2, 4, 5, 8, 9, 11, 12$. F_i are indicated by the shaded areas in Fig. 3(c). We carry out the RRQR on F_i:

$$F_i \approx \widetilde{F}_i V_i^H, \tag{4.10}$$

where $\widetilde{F}_i$ can be denoted as $A|_{\hat{t}_i \times \tilde{t}_i}$ and $\tilde{t}_i$ is the update of t_i due to the rank contraction in the column compression. We note that this step is done locally within each process. One of the *HSS generators* V_i is obtained here.

To enable the upper level column HSS construction, communications occur pairwise: $\{0\} \leftrightarrow \{1\}$, $\{2\} \leftrightarrow \{3\}$, $\{4\} \leftrightarrow \{5\}$, $\{6\} \leftrightarrow \{7\}$. The upper level off-diagonal blocks $F_i, i = 3, 6, 10, 13$ are formed by merging $\widetilde{F}_{c_1}$ and $\widetilde{F}_{c_2}$ via the following formula:

$$F_i = \begin{pmatrix} A|_{\hat{t}_i \times \tilde{t}_{c_1}}, & A|_{\hat{t}_i \times \tilde{t}_{c_2}} \end{pmatrix}, \qquad B_{c_1} = A|^H_{\tilde{t}_{c_1} \times \tilde{t}_{c_2}}, \qquad B_{c_2} = A|^H_{\tilde{t}_{c_2} \times \tilde{t}_{c_1}}. \tag{4.11}$$

This procedure is illustrated in Fig. 3(d). Two communication steps are needed: in the first step B_{c_1} and B_{c_2} are generated by exchanging $A|_{\tilde{t}_{c_2} \times \tilde{t}_{c_1}}$ and $A|_{\tilde{t}_{c_1} \times \tilde{t}_{c_2}}$ pairwise between c_1 context and c_2 context. We note that one of *HSS generators* B_i is obtained here. The second stage is to redistribute the newly merged off-diagonal block F_i onto the process grid associated with the node i's context for $i = 3, 6, 10, 13$.

Column compression—Step 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 $F_i, i = 3, 6, 10, 13$ has already been distributed onto the respective process context, as illustrated in Fig. 3(d). Then we perform parallel RRQR (see Section 4.1) for each F_i:

$$F_i = \widetilde{F}_i \begin{pmatrix} W_{c_1}^H, & W_{c_2}^H \end{pmatrix}, \tag{4.12}$$

where $\widetilde{F}_i$ can also be denoted as $A|_{\hat{t}_i \times \tilde{t}_i}$ and $\tilde{t}_i$ is the column index subset of $\widetilde{F}_i$. One of the *HSS generators* W_i is generated here.

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 Fig. 3(e). Similar to Eqn.(4.11), two communication steps are needed.

Column compression—Step 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 Fig. 3(e). Then we perform RRQR similar to Eqn.(4.12). One of the *HSS generators* W_i is also obtained here.

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 Eqn.(4.11) except there is no merging step to form F_{15}. Fig. 3(f) indicates that after this final communication, there are no residual off-diagonal blocks. All the *HSS generators* $D_i, U_i, R_i, B_i, W_i, V_i$ have been successfully computed.

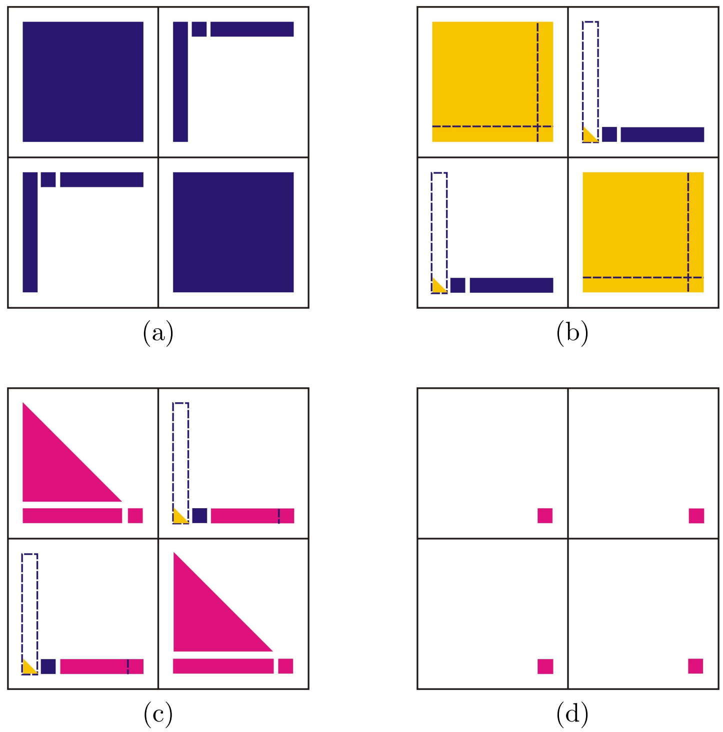

FIG. 4. *(a). the illustration of a block 2×2 HSS form for (5.1); (b). the illustration of QL factorization for (5.2); (c). the illustration of LQ factorization for (5.3); (d). the illustration of the inter-context communication to form (5.5).*

5. Parallel HSS factorization. After the HSS representation of A (2.1) is constructed in parallel, it is ready to factorize A via the *HSS generators*. Here we adopt the ULV-type factorization and generalize our discussions by a block 2×2 HSS form illustrated by Figure 4(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 the generators with subscripts c_1 are distributed on the process grid associated with c_1 context, the generators with subscripts c_2 are distributed on the process grid associated with c_2 context, and the generators with subscripts i are distributed on the process grid associated with i context. c_1 and c_2 are the children of i on the HSS tree $\mathcal{T}$. The i context is the union of the c_1 context and the c_2 context. We assume that the size of U_{c_1} is $m_1 \times r_1$, and the size of U_{c_2} is $m_2 \times r_2$. Here r_1 and r_2 are numerical ranks.

We start with the QL factorization of U_{c_1} and U_{c_2}, which is illustrated by Figure 4(b):

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

where $\widetilde{U}_{c_1}$ and $\widetilde{U}_{c_2}$ are lower triangular matrices of the size $r_1 \times r_1$ and $r_2 \times r_2$, respectively. We note that there is no inter-context communication occurring at this stage. Then we multiply $Q_{c_1}^H$ and $Q_{c_2}^H$ independently within each context and obtain:

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

where

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

in which $\widehat{D}_{c_1;2,2}$ and $\widehat{D}_{c_2;2,2}$ are of the size $r_1 \times r_1$ and $r_2 \times r_2$, respectively.

Then an LQ factorization is carried out independently within each context:

$$\begin{aligned} \begin{pmatrix} \widehat{D}_{c_1;1,1} & \widehat{D}_{c_1;1,2} \end{pmatrix} &= \begin{pmatrix} \widetilde{D}_{c_1;1,1} & 0 \end{pmatrix} P_{c_1}, \\ \begin{pmatrix} \widehat{D}_{c_2;1,1} & \widehat{D}_{c_2;1,2} \end{pmatrix} &= \begin{pmatrix} \widetilde{D}_{c_2;1,1} & 0 \end{pmatrix} P_{c_2}. \end{aligned} \tag{5.3}$$

We multiply P_{c_1} and P_{c_2} independently within each context, which can be illustrated by Figure 4(c), and obtain:

$$\begin{aligned} &\begin{pmatrix} Q_{c_1}^H & 0 \\ 0 & Q_{c_2}^H \end{pmatrix} \begin{pmatrix} D_{c_1} & U_{c_1} B_{c_1} V_{c_2}^H \\ U_{c_2} B_{c_2} V_{c_1}^H & D_{c_2} \end{pmatrix} \begin{pmatrix} P_{c_1}^H & 0 \\ 0 & P_{c_2}^H \end{pmatrix} \\ &= \begin{pmatrix} \begin{pmatrix} \widetilde{D}_{c_1;1,1} & 0 \\ \widetilde{D}_{c_1;2,1} & \widetilde{D}_{c_1;2,2} \end{pmatrix} & \begin{pmatrix} 0 \\ \widetilde{U}_{c_1} B_{c_1} \begin{pmatrix} \widetilde{V}_{c_2;1}^H & \widetilde{V}_{c_2;2}^H \end{pmatrix} \end{pmatrix} \\ \begin{pmatrix} 0 \\ \widetilde{U}_{c_2} B_{c_2} \begin{pmatrix} \widetilde{V}_{c_1;1}^H & \widetilde{V}_{c_1;2}^H \end{pmatrix} \end{pmatrix} & \begin{pmatrix} \widetilde{D}_{c_2;1,1} & 0 \\ \widetilde{D}_{c_2;2,1} & \widetilde{D}_{c_2;2,2} \end{pmatrix} \end{pmatrix}, \end{aligned} \tag{5.4}$$

where

$$\begin{aligned} \begin{pmatrix} \widetilde{D}_{c_1;2,1} & \widetilde{D}_{c_1;2,2} \end{pmatrix} &= \begin{pmatrix} \widehat{D}_{c_1;2,1} & \widehat{D}_{c_1;2,2} \end{pmatrix} P_{c_1}^H, \\ \begin{pmatrix} \widetilde{D}_{c_2;2,1} & \widetilde{D}_{c_2;2,2} \end{pmatrix} &= \begin{pmatrix} \widehat{D}_{c_2;2,1} & \widehat{D}_{c_2;2,2} \end{pmatrix} P_{c_2}^H, \\ \begin{pmatrix} \widetilde{V}_{c_1;1}^H & \widetilde{V}_{c_1;2}^H \end{pmatrix} &= V_{c_1}^H P_{c_1}^H, \\ \begin{pmatrix} \widetilde{V}_{c_2;1}^H & \widetilde{V}_{c_2;2}^H \end{pmatrix} &= V_{c_2}^H P_{c_2}^H. \end{aligned}$$

We note that there is still no inter-context communication occurring up to this stage.

Eventually we form the parent D_i, U_i and V_i recursively via inter-context communications, which is illustrated by Figure 4(d):

$$D_i = \begin{pmatrix} \widetilde{D}_{c_1;2,2} & \widetilde{U}_{c_1} B_{c_1} \widetilde{V}_{c_2;2}^H \\ \widetilde{U}_{c_2} B_{c_2} \widetilde{V}_{c_1;2}^H & \widetilde{D}_{c_2;2,2} \end{pmatrix}, \quad U_i = \begin{pmatrix} \widetilde{U}_{c_1} R_{c_1} \\ \widetilde{U}_{c_2} R_{c_2} \end{pmatrix}, \quad V_i = \begin{pmatrix} \widetilde{V}_{c_1;2} W_{c_1} \\ \widetilde{V}_{c_2;2} W_{c_2} \end{pmatrix}. \tag{5.5}$$

Eq.(5.5) maintains the form of the recursive definition (2.1) of the HSS generators, except the size has been deducted due to the HSS construction introduced in section 4.

If the root node is reached, an LU factorization with partial pivoting is conducted on D_i.

6. Parallel HSS solution. We solve the linear system of equations $Ax = b$ after obtaining the HSS form of A in section 4 and the HSS factorization in section 5. We stick to the convention of a block 2×2 HSS form of A adopted in section 5, to generalize our discussion.

The system $Ax = b$ with the HSS representation of A can be written in the following equation:

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

where

$$x = \begin{pmatrix} x_{c_1} \\ x_{c_2} \end{pmatrix}, \qquad b = \begin{pmatrix} b_{c_1} \\ b_{c_2} \end{pmatrix}.$$

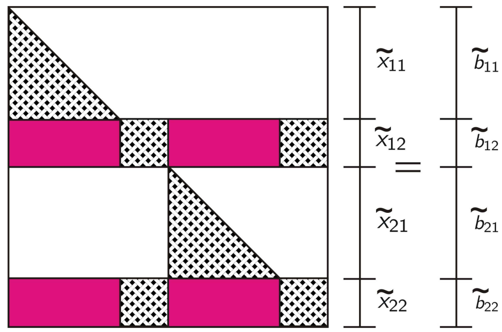

FIG. 5. *The illustration of the linear system of equations (6.2).*

With the aid of eq.(5.4), we can rewrite eq.(6.1) into the following form:

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

where

$$x_{c_1} = P^H_{c_1} \widetilde{x}_{c_1} = P^H_{c_1} \begin{pmatrix} \widetilde{x}_{c_1;1} \\ \widetilde{x}_{c_1;2} \end{pmatrix}, \qquad x_{c_2} = P^H_{c_2} \widetilde{x}_{c_2} = P^H_{c_2} \begin{pmatrix} \widetilde{x}_{c_2;1} \\ \widetilde{x}_{c_2;2} \end{pmatrix};$$
$$b_{c_1} = Q_{c_1} \widetilde{b}_{c_1} = Q_{c_1} \begin{pmatrix} \widetilde{b}_{c_1;1} \\ \widetilde{b}_{c_1;2} \end{pmatrix}, \qquad b_{c_2} = Q_{c_2} \widetilde{b}_{c_2} = Q_{c_2} \begin{pmatrix} \widetilde{b}_{c_2;1} \\ \widetilde{b}_{c_2;2} \end{pmatrix}.$$

Figure 5 illustrates the eq.(6.2). We point out that the solution to eq.(6.1) is converted to the solution to eq.(6.2). Once $\widetilde{x}_{c_1}$ and $\widetilde{x}_{c_2}$ are obtained, we can easily compute the original solution x.

We note that the following two triangular systems can be efficiently solved locally within each context:

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

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

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

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

Then we have to further update the right hand side via inter-context communication:

$$\begin{aligned} \widetilde{b}_{c_1;2} &= \widetilde{b}_{c_1;2} - \widetilde{U}_{c_1} B_{c_1} \widetilde{V}^H_{c_2;1}\ \widetilde{x}_{c_2;1} \\ \widetilde{b}_{c_2;2} &= \widetilde{b}_{c_2;2} - \widetilde{U}_{c_2} B_{c_2} \widetilde{V}^H_{c_1;1}\ \widetilde{x}_{c_1;1} \end{aligned}$$

Eventually we solve two triangular systems on the i context:

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

N (2D mesh $N \times N$)	$5,000$	$10,000$	$20,000$	$40,000$	$80,000$
size ($\times 10^9$)	0.025	0.1	0.4	1.6	6.4
Nprocs	16	64	256	1024	4096
MF + HSS (s)	89	175	366	780	1689
last frontal size	5,000	10,000	20,000	40,000	80,000
HSS compr (s)	2.09	3.89	6.59	17.94	47.77
RRQR (s)	1.81	2.86	4.53	12.19	32.39
redist (s)	0.18	1.03	2.06	5.75	15.38

TABLE 7.1
Weak scaling of the parallel HSS solver imbedded in the parallel multifrontal solver, for 2D Helmholtz problem.

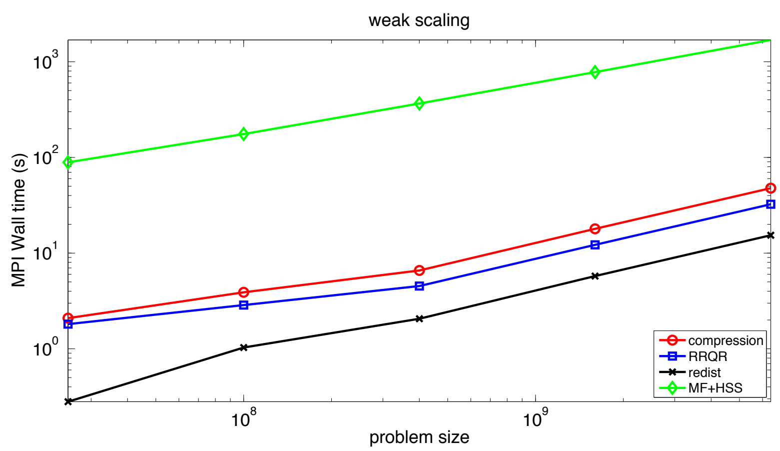

FIG. 6. *Weak scaling curves for HSS compression, RRQR phases in HSS compression, data redistribution phases in HSS compression, and the entire parallel multifrontal solver equipped with the parallel HSS solver. The data is from Table 7.1.*

7. Performance tests and numerical examples. We first present the weak scaling results for both the parallel HSS solver, and the hybrid parallel multifrontal solver in which the parallel HSS solver is imbedded, for solving large scale 2D Helmholtz problem. The 2D mesh ranges from $10,000 \times 10,000$ to $80,000 \times 80,000$, yielding that the size of the global Helmholtz matrix ranges from 0.1 billion up to 6.4 billion. We carry out our experiments on the cluster `Hopper` at Lawrence Berkeley National Laboratory (LBNL) `hopper.nersc.gov`. In order to test the weak scaling, we make the number of processors four times larger upon doubling the mesh size. We have 8 cores per node (32GB memeory). The MPI wall time for the entire hybrid solver is recorded in Table 7.1. To highlight the scalability of the parallel HSS solver, we focus on the last frontal dense matrix computation in the multifrontal solver. We split the total HSS compression time into an RRQR phase and a data redistribution phase. Table 7.1 also lists the MPI wall time for each individual phase. The weak scaling curve is plotted in Figure 6. We note that four phases in the entire hybrid solver scale much the same. The weak scaling factor is about 2.0.

Secondly, we show the accuracy of the entire hybrid parallel multifrontal and HSS solver for seismic applications. The mesh size is 5000×3000. Figure 7(a) displays a 5Hz time-harmonic wavefield solution to the 2D Helmholtz problem using the hybrid parallel multifrontal and HSS solver, with the preset tolerance $\tau = 10^{-2}$. The amplitude difference between the solution in Figure 7(a) and the true solution is displayed in Figure 7(b). We note that 2 digits is insufficient to produce an accepTable wavefield solution, since too much information is lost when RRQR is conducted with a large tolerance. In Figure 7(c) we display the 5Hz time-harmonic wavefield solution to the 2D Helmholtz problem using the hybrid parallel multifrontal and HSS solver with the preset tolerance

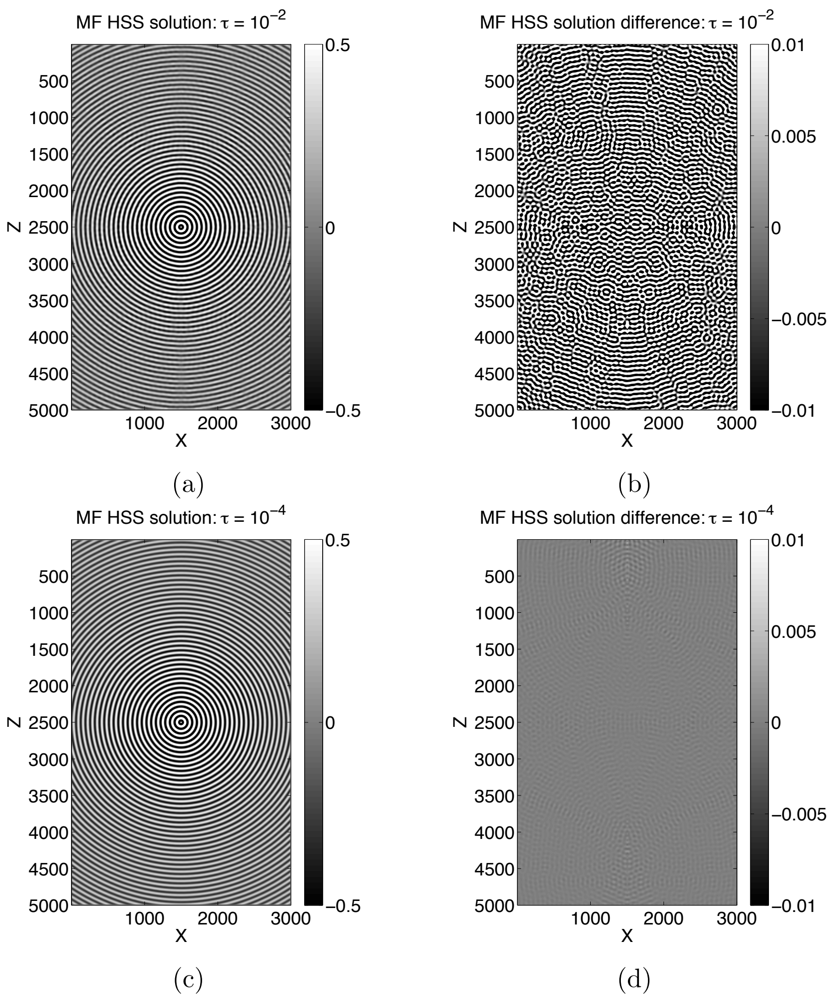

FIG. 7. *(a). 5Hz time-harmonic wavefield solution to the 2D Helmholtz problem using the hybrid parallel multifrontal and HSS solver, with the preset tolerance $\tau = 10^{-2}$; (b). the amplitude difference between (a) and the true solution; (c). 5Hz time-harmonic wavefield solution to the 2D Helmholtz problem using the hybrid parallel multifrontal and HSS solver, with the preset tolerance $\tau = 10^{-4}$; (d). the amplitude difference between (c) and the true solution.*

Nprocs	64	128	256	512	1024
HSS compr (s)	111	61	38	33	36
RRQR (s)	83	45	28	26	23
redist (s)	28	16	10	7	13

TABLE 7.2
Strong scaling for a fixed Toeplitz matrix $100,000 \times 100,000$.

$\tau = 10^{-4}$. The amplitude difference is displayed in Figure 7(d); 4 digits yields an accuracy typically required in seismic applications.

Finally, we present the strong scaling in Table 7.2 of the parallel HSS compression for a fixed dense $100,000 \times 100,000$ Toeplitz matrix. The strong scaling curve is plotted in Figure 8. We note that RRQR, compared with the data redistribution, is the most time consuming phase in the parallel HSS compression.

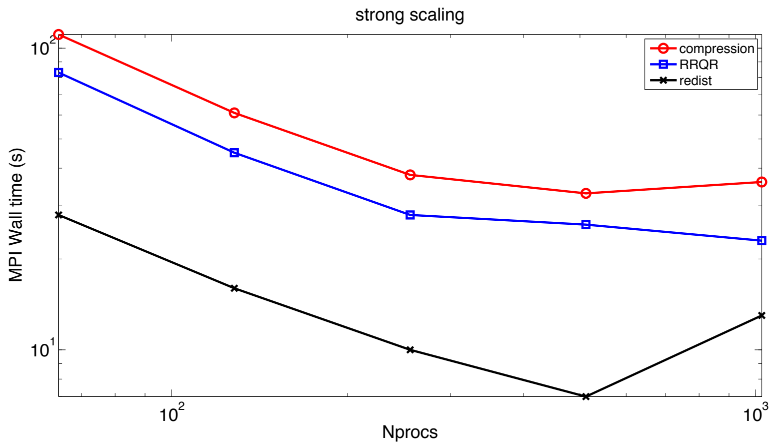

Fig. 8. *Strong scaling curve for HSS compression, RRQR phase and the data redistribution phase, for a dense* $100,000 \times 100,000$ *Toeplitz matrix. The data is from Table 7.2.*

8. Conclusions. We presented fast parallel Hierarchically SemiSeparable matrix (HSS) algorithms: parallel HSS construction using the rank revealing QR (RRQR) method, parallel HSS ULV factorization, and parallel solution. We studied their scalability. We exploited the portability of two libraries, `BLACS` and `ScaLapack`. Computational examples of weak scaling, strong scaling and accuracy demonstrated that our implementation is robust and efficient indeed.

9. Acknowledgements. We thank the members, ConocoPhillips, ExxonMobil, PGS, Statoil and Total, of the Geo-Mathematical Imaging Group (GMIG) for 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.

REFERENCES

[1] `www.netlib.org/blacs`.
[2] `www.netlib.org/lapack`.
[3] `www.netlib.org/scalapack`.
[4] T. Bella, Y. Eidelman, and V. Gohberg, I. and. Olshevsky, *Computations with quasiseparable polynomials and matrices*, Theoret. Comput. Sci., 409 (2008), pp. 158–179.
[5] S. Börm, L. Grasedyck, and W. Hackbusch, *Introduction to hierarchical matrices with applications*, Eng. Anal. Bound. Elem, 27 (2003), pp. 405–422.
[6] 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).
[7] 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.
[8] S. Delvaux and M. Van Barel, *A QR-based solver for rank structured matrices*, SIAM. J. Matrix Anal. Appl., 30 (2008), pp. 464–490.
[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*, Integral Equations Operator Theory, 34 (1999), pp. 293–324.
[11] W. Hackbusch, *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: introduction to $\mathcal{H}$-matrices*, Computing, 62 (1999), pp. 89–108.
[12] 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.
[13] J.W.H. Liu, *The multifrontal method for sparse matrix solution: Theory and practice*, SIAM Review, 34 (1992), pp. 82–109.
[14] R. Vandebril, M. Van Barel, G. Golub, and N. Mastronardi, *A bibliography on semiseparable matrices*, Calcolo, 42 (2005), pp. 249–270.

[15] B. Khoromskij W. Hackbusch and S. A. Sauter, *On $\mathcal{H}^2$-matrices*, Lectures on applied mathematics (Munich, 1999), Springer, Berlin, (2000), pp. 9–29.
[16] L. Grasedyck W. Hackbusch and S. Börm, *An introduction to hierarchical matrices*, Math. Bohem., 127 (2002), pp. 229–241.
[17] 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).
[18] 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.
[19] ———, *Fast algorithms for hierarchically semiseparable matrices*, Numer. Linear Algebra Appl., 2010 (2010), pp. 953–976.