Research paper

International Journal of HIGH PERFORMANCE COMPUTING APPLICATIONS

The International Journal of High Performance Computing Applications
1–15
© The Author(s) 2020
Article reuse guidelines:
sagepub.com/journals-permissions
DOI: 10.1177/1094342020918305
journals.sagepub.com/home/hpc
SAGE

A parallel hierarchical blocked adaptive cross approximation algorithm

Yang Liu[1], Wissam Sid-Lakhdar[1], Elizaveta Rebrova[2], Pieter Ghysels[1] and Xiaoye Sherry Li[1]

Abstract

This article presents a low-rank decomposition algorithm based on subsampling of matrix entries. The proposed algorithm first computes rank-revealing decompositions of submatrices with a blocked adaptive cross approximation (BACA) algorithm, and then applies a hierarchical merge operation via truncated singular value decompositions (H-BACA). The proposed algorithm significantly improves the convergence of the baseline ACA algorithm and achieves reduced computational complexity compared to the traditional decompositions such as rank-revealing QR. Numerical results demonstrate the efficiency, accuracy, and parallel scalability of the proposed algorithm.

Keywords

Adaptive cross approximation, singular value decomposition, rank-revealing decomposition, parallelization, multilevel algorithms

1. Introduction

Rank-revealing decomposition algorithms are important numerical linear algebra tools for compressing high-dimensional data, accelerating solution of integral and partial differential equations, constructing efficient machine learning algorithms, analyzing numerical algorithms, and so on, as matrices arising from many science and engineering applications oftentimes exhibit numerical rank-deficiency. Despite the favorable $O(nr)$ memory footprint of such decompositions with n and r, respectively, denoting the matrix dimension (assuming a square matrix) and the numerical rank, the computational cost can be expensive. Existing rank-revealing decompositions such as truncated singular value decomposition (SVD), column-pivoted QR (QRCP), CUR decomposition, interpolative decomposition (ID), and rank-revealing LU typically require at least $O(n^2r)$ operations (Cheng et al., 2005; Gu and Eisenstat, 1996; Mahoney and Drineas, 2009; Voronin and Martinsson, 2017). This complexity can be reduced to $O(n^2 \log r + nr^2)$ by structured random matrix projection-based algorithms (Liberty et al., 2007; Voronin and Martinsson, 2017). In addition, faster algorithms are available in the following three scenarios.

1. When each element entry can be computed in $O(1)$ CPU time with prior knowledge (i.e. smoothness, sparsity, or leverage scores) about the matrix, faster algorithms such as randomized CUR and adaptive cross approximation (ACA) (Bebendorf, 2000; Bebendorf and Grzhibovskis, 2006; Zhao et al., 2005) algorithms can achieve $O(nr^2)$ complexity. However, the robustness of these algorithms relies heavily on matrix properties that are not always present in practice.
2. When the matrix can be rapidly applied to arbitrary vectors, algorithms such as randomized SVD, QR, and UTV (T lower or upper triangular) (Feng et al., 2019; Liberty et al., 2007; Martinsson et al., 2019; Xiao et al., 2017) can be utilized to achieve quasi-linear complexity.
3. Finally, given a matrix with missing entries, the low-rank decomposition can be constructed via matrix completion algorithms (Balzano et al., 2010; Candès and Recht, 2009) in quasi-linear time assuming incoherence properties of the matrices (i.e. projection of natural basis vectors onto the space spanned by singular vectors of the matrix should not be very sparse).

[1] Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA, USA
[2] Department of Mathematics, University of California, Los Angeles, CA, USA

Corresponding author:
Yang Liu, Computational Research Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA.
Email: liuyangzhuan@lbl.gov

This work concerns the development of a practical algorithm, in application scenario 1, that improves the robustness of ACA algorithms while maintaining reduced complexity for broad classes of matrices.

The partially pivoted ACA algorithm, closely related to LU with rook pivoting (Foster, 1997; Neal and Poole, 1992; Poole and Neal, 1992, 1991), constructs an LU-type decomposition upon accessing one row and column per iteration. For matrices resulting from asymptotically smooth kernels, ACA is a rank-revealing and optimal-complexity algorithm that converges in $O(r)$ iterations (Bebendorf, 2000). Despite its favorable computational complexity, it is well-known that the ACA algorithm suffers from deteriorated convergence and/or premature termination for non-smooth, sparse, and/or coherent matrices (Heldring et al., 2014). Hybrid methods or improved convergence criteria (e.g. hybrid ACA-CUR, averaging, statistical norm estimation) have been proposed to partially alleviate the problem (Grasedyck and Hackbusch, 2005; Heldring et al., 2015). The main difficulty of leveraging ACA as robust algebraic tools for general low-rank matrices results from ACA's partial pivot-search strategy to attain low complexity. In addition to the above-mentioned remedies, another possibility to improve ACA's robustness is to search for pivots in a wider range of rows/columns without sacrificing too much computational efficiency. Here we consider two different strategies:

1. Instead of searching one row/column per iteration as in ACA, it is possible to search a block of rows/columns to find multiple pivots together.
2. Instead of applying ACA directly on the entire matrix, it is possible to start with compressing submatrices via ACA and then merge the results as one low-rank product.

In extreme cases (e.g. when block size equals matrix dimension or submatrix dimension equals one), these strategies lead to quadratic computational costs. Therefore, it is valuable to address the question: for what matrix kernels and under what block/submatrix sizes will these strategies retain low complexity.

For the first strategy, this work proposes a blocked ACA algorithm (BACA) that extracts a block row/column per iteration to significantly improve convergence of the baseline ACA algorithms. The blocked version also enjoys higher flop performance as it involves mainly BLAS-3 operations. Compared to the aforementioned remedies, the proposed algorithm provides a unified framework to balance robustness and efficiency. Upon increasing the block size (i.e. the number of rows/columns per iteration), the algorithm gradually changes from ACA to ID. For the second strategy, the proposed algorithm further subdivides the matrix into n_b submatrices compressed via BACA, followed by a hierarchical merge algorithm leveraging low-rank arithmetic (Grasedyck and Hackbusch, 2003; Hackbusch et al., 2002). The overall cost of this H-BACA algorithm is at most $O(\sqrt{n_b}nr^2)$ assuming the block size in BACA is less than the rank. In other words, the proposed H-BACA algorithm is a general numerical linear algebra tool as an alternative to ACA, SVD, QR, and so on. In addition, the overall algorithm can be parallelized using distributed-memory linear algebra packages such as ScaLAPACK (Blackford et al., 1997) which avoids the difficulty of efficient parallelization of plain ACA algorithms. Numerical results illustrate good accuracy, efficiency, and parallel performance. In addition, the proposed algorithm can be used as a general low-rank compression tool for constructing hierarchical matrices (Rebrova et al., 2018).

2. Notation

Throughout this article, we adopt the Matlab notation of matrices and vectors. Submatrices of a matrix A are denoted $A(I, J)$, $A(:, J)$, or $A(I, :)$ where I and J are index sets. Similarly, subvectors of a column vector u are denoted $u(I)$. An index set I permuted by J reads $I(J)$. Transpose, inverse, pseudo-inverse of A are $A^t, A^{-1}, A^{\dagger}$. $||A||_F$ and $||u||_2$ denote Frobenius norm and 2-norm. Note that u refers to a $n \times 1$ column vector. Vertical and horizontal concatenations of A, B are $[A; B]$ and $[A, B]$. Element-wise multiplication of A and B is $A \circ B$. All matrices are real-valued unless otherwise stated. It is assumed for $A \in \mathbb{R}^{m\times n}$, $m = O(n)$, but the proposed algorithms also apply to complex-valued and tall-skinny/short-fat matrices. We denote truncated SVD as $[U, \Sigma, V, r] = SVD(A, \varepsilon)$ with $U \in \mathbb{R}^{m\times r}$, $V^t \in \mathbb{R}^{n\times r}$ column orthogonal, $\Sigma \in \mathbb{R}^{r\times r}$ diagonal, and r being ε-rank defined by $r = \min\{k \in \mathbb{N} : \Sigma_{k+1,k+1} < \varepsilon\Sigma_{1,1}\}$. We denote QRCP as $[Q, T, J] = QR(A, r)$ or $[Q, T, J] = QR(A, \varepsilon)$ with $Q \in \mathbb{R}^{m\times r}$ column orthogonal, $T \in \mathbb{R}^{r\times n}$ upper triangular, J being column pivots, and ε and r being the prescribed accuracy and rank, respectively. QR without column-pivoting is simply written as $[Q, T] = QR(A)$. Cholesky decomposition without pivoting is written as $T = Chol(A)$ with T upper triangular. $\log n$ means logarithm of n to the base 2.

3. Algorithm description

3.1. Adaptive cross approximation

Before describing the proposed algorithm, we first briefly summarize the baseline ACA algorithm (Zhao et al., 2005). Consider a matrix $A \in \mathbb{R}^{m\times n}$ of ε-rank r, the ACA algorithm approximates A by a sequence of rank-1 outer products as

$$A \approx UV = \sum_{k=1}^{r} u_k v_k^t \tag{1}$$

Define the residual matrix E_k as

$$E_0 = A, \quad E_k = E_{k-1} - u_k v_k^t. \tag{2}$$

At each iteration k, the algorithm selects column u_k (pivot j_k from remaining columns) and row v_k^t (pivot i_k from remaining rows) from the residual matrix E_{k-1}

Algorithm 1. Adaptive cross approximation algorithm (ACA).

```
input : Matrix $A \in \mathbb{R}^{m \times n}$, relative tolerance $\epsilon$
output: Low-rank approximation of $A \approx UV$ with
        rank $r$
1  $U = 0, V = 0, \mu = 0, r_0 = 0$, $j_1$ is a random
   column index;
2  for $k = 1$ to $\min\{m, n\}$ do
3      $u_k = E_{k-1}(:, j_k) = A(:, j_k) - UV(:, j_k)$;
4      $i_k = \arg\max_i |u_k(i)|$;
5      $u_k \leftarrow u_k / u_k(i_k)$;
6      $v_k^t = E_{k-1}(i_k, :) = A(i_k, :) - U(i_k, :)V$;
7      $j_{k+1} = \arg\max_j |v_k(j)|$;
8      $\nu^2 = \|u_k\|_2^2 \|v_k\|_2^2$;
9      $\mu^2 \leftarrow \mu^2 + \nu^2 + 2(u_k^t U)(V v_k)$;
10     $U \leftarrow [U, u_k], V \leftarrow [V; v_k^t], r_k = r_{k-1} + 1$;
11     Terminate if $\nu < \epsilon\mu$.
```

corresponding to an element denoted by $E_{k-1}(i_k, j_k)$ with sufficiently large magnitude. Note that u_k and v_k are $m \times 1$ and $n \times 1$ vectors. The partially pivoted ACA algorithm (ACA for short), selecting j_k, i_k by only looking at previously selected rows and columns, is described as Algorithm 1. Specifically, each iteration k selects pivot i_k used in the current iteration and pivot j_{k+1} for the next iteration (via lines 4 and 7) as

$$i_k = \underset{i \neq i_1, \ldots, i_{k-1}}{\arg\max} |E_{k-1}(:, j_k)| \tag{3}$$

$$j_{k+1} = \underset{j \neq j_1, \ldots, j_k}{\arg\max} |E_{k-1}(i_k, :)| \tag{4}$$

and j_1 is a random initial column index. Note that $i_k \neq i_1, \ldots, i_{k-1}$ and $j_k \neq j_1, \ldots, j_{k-1}$ are enforced. The iteration is terminated when $\nu < \varepsilon\mu$ with

$$\nu = ||u_k v_k^t||_F \approx ||E_k||_F, \quad \mu = ||UV||_F \approx ||A||_F \tag{5}$$

and ε is the prescribed tolerance. Note that each iteration requires only $O(nr_k)$ flop operations with r_k denoting currently revealed numerical rank. The overall complexity of partially pivoted ACA scales as $O(nr^2)$ when the algorithm converges in $O(r)$ iterations. Despite the favorable complexity, the convergence of ACA for general rank-deficient matrices is unsatisfactory. For many rank-deficient matrices arising from the numerical solution of PDEs, signal processing and data science, ACA oftentimes either requires $O(n)$ iterations or exhibits premature termination. First, as ACA does not search the full residual matrices for the largest element, it cannot avoid selection of smaller pivots for general rank-deficient matrices and may require $O(n)$ iterations. Second, the approximation $||u_k v_k^t||_F$ in (5) often causes the premature termination with the selection of smaller pivots. Remedies such as averaged stopping criteria (Zhou et al., 2017), stochastic error estimation (Heldring et al., 2015), ACA+ (Grasedyck and Hackbusch, 2005), and hybrid ACA (Grasedyck and Hackbusch, 2005) have been developed but they do not generalize to a broad range of applications.

3.2. Blocked adaptive cross approximation

Instead of selecting only one column and row from the residual matrix in each ACA iteration, we can select a fixed-size block of columns and rows per iteration to improve the convergence and accuracy of ACA. In addition, many BLAS-1 and BLAS-2 operations of ACA become BLAS-3 operations and hence higher flop performance can be achieved.

Specifically, the proposed BACA algorithm factorizes A

$$A \approx UV = \sum_{k=1}^{n_d} U_k V_k \tag{6}$$

where $U_k \in \mathbb{R}^{m \times d_k}$ and $V_k \in \mathbb{R}^{d_k \times n}$. In principle, the algorithm selects a block of d rows and columns via cross approximations in the residual matrix and then $d_k \leq d$ ones via rank-revealing algorithms to form a low-rank update at iteration k. The total number of iterations is approximately $n_d \approx r/d$ if $d_k \approx d$. Instead of selecting row/column pivots via lines 4 and 7 of Algorithm 1, the proposed algorithm selects row and column index sets I_k and J_k by performing QRCP on d columns (more precisely their transpose) and rows of the residual matrices. This proposed strategy is described in Algorithm 2.

Each BACA iteration is composed of three steps.

- Find block row I_k and block column J_{k+1} by QRCP. Starting with a random column index set J_1, the block row I_k and the next iteration's block column J_{k+1} are selected by (lines 4 and 7)

$$[Q_k^c, T_k^c, I_k] = QR(E_{k-1}(:, J_k)^t, d) \tag{7}$$

$$[Q_{k+1}^r, T_{k+1}^r, I_{k+1}] = QR(E_{k-1}(I_k, :), d) \tag{8}$$

Here the algorithm first selects d skeleton rows from the submatrix $E_{k-1}(:, J_k)$ (i.e. d columns from its transpose) and then selects d skeleton columns from the submatrix $E_{k-1}(I_k, :)$ by leveraging the LAPACK implementation of QRCP as it provides a simple way of greedily selecting well-conditioned columns by examining column norms in the R factor at each iteration. Note that many other subset selection algorithms exist in both the machine learning and numerical linear algebra communities (e.g. strong rank-revealing QR (Gu and Eisenstat, 1996), spectrum-revealing QR (Feng et al., 2019), and column subset selection problems (Boutsidis et al., 2009)), which ideally pick d matrix columns with maximum volumes. Note that I_k excludes rows selected in previous iterations. To efficiently enforce such condition, the QRCP is performed on the submatrix of $E_{k-1}(:, J_k)^t$ excluding previously selected rows rather than directly on $E_{k-1}(:, J_k)^t$. Similarly, J_k excludes columns selected in previous iterations. See Figure 1(a) for an illustration of the procedure. I_k and J_{k+1} are selected by QRCP on the column

Algorithm 2. Blocked adaptive cross approximation (BACA) algorithm.

```
   input : Matrix $A \in \mathbb{R}^{m\times n}$, block size $d$, relative
           tolerance $\epsilon$
   output: Low-rank approximation of $A \approx UV$ with
           rank $r$
1  $U = 0, V = 0, r_0 = 0, \mu = 0, \bar{J}_1$ is a random
     index set of cardinality $d$;
2  for $k = 1$ to $\min\{m, n\}$ do
3      $C_k = E_{k-1}(:, J_k) = A(:, J_k) - UV(:, J_k)$;
4      $[Q^c_k, T^c_k, I_k] = \texttt{QR}(C^t_k, d)$, $I_k$ denotes selected
       skeleton rows;
5      $R_k = E_{k-1}(I_k, :) = A(I_k, :) - U(I_k, :)V$;
6      $[Q^r_{k+1}, T^r_{k+1}, J_{k+1}] = \texttt{QR}(R_k, d)$, $J_{k+1}$
       denotes selected skeleton columns;
7      $W_k = E_{k-1}(I_k, J_k) = A(I_k, J_k) - U(I_k, :)V(:, J_k)$;
8      $[U_k, V_k, d_k, \bar{J}] = \texttt{LRID}(C_k, W_k, R_k, \epsilon)$;
9      $I_k \leftarrow I_k([1:d_k]), J_k \leftarrow J_k(\bar{J})$;
10     $r_k = r_{k-1} + d_k$;
11     $\nu = \texttt{LRnorm}(U_k, V_k)$;
12     $\mu \leftarrow \texttt{LRnormUp}(U, V, \mu, U_k, V_k, \nu)$;
13     $U \leftarrow [U, U_k], V \leftarrow [V; V_k]$;
14     Terminate if $\nu < \epsilon\mu$.
15 Function LRID ($C, W, R, \epsilon$)
       input : $C = A(:, J), R = A(I, :)$,
               $W = A(I, J)$ with $I, J$ of same
               cardinality
       output: $A \approx UV$ with $U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{r\times n}$
16     $[Q, T, \bar{J}, r] = \texttt{QR}(W, \epsilon)$;
17     $U = C(:, \bar{J})$;
18     $V = T^{-1}Q^t R$;
19     return $U, V, r, \bar{J}$
20 Function LRnorm ($U, V$)
       input : $A = UV$
       output: $\|A\|_F$
21     $T_1 = \texttt{Chol}(U^t U)$;
22     $T_2 = \texttt{Chol}(V V^t)$;
23     return $\|T_1 T_2^t\|_F$;
24 Function LRnormUp ($U, V, \nu, \bar{U}, \bar{V}, \bar{\nu}$)
       input : $U \in \mathbb{R}^{m\times r}, V \in \mathbb{R}^{r\times n}, \bar{U} \in \mathbb{R}^{m\times \bar{r}}$,
               $\bar{V} \in \mathbb{R}^{\bar{r}\times n}, \nu = \|UV\|_F, \bar{\nu} = \|\bar{U}\bar{V}\|_F$
       output: $\|[U, \bar{U}][V; \bar{V}]\|_F$
25     $s = \nu^2 + \bar{\nu}^2 + 2\sum_{i=1}^{r}\sum_{j=1}^{\bar{r}} \tilde{V}(i, j)$ with
       $\tilde{V} = (V\bar{V}^t) \circ (U^t\bar{U})$;
26     return $\sqrt{s}$
```

and transpose of the row marked in yellow, respectively. The column marked in grey is used to select I_{k+1} in the next iteration. For illustration purpose, index sets in Figure 1(a) consist of contiguous indices.

- Form the factors of the low-rank product $U_k V_k$. Let $C_k = E_{k-1}(:, J_k)$, $R_k = E_{k-1}(I_k, :)$, and $W_k = E_{k-1}(I_k, J_k)$, E_{k-1} can be approximated by an ID-type decomposition

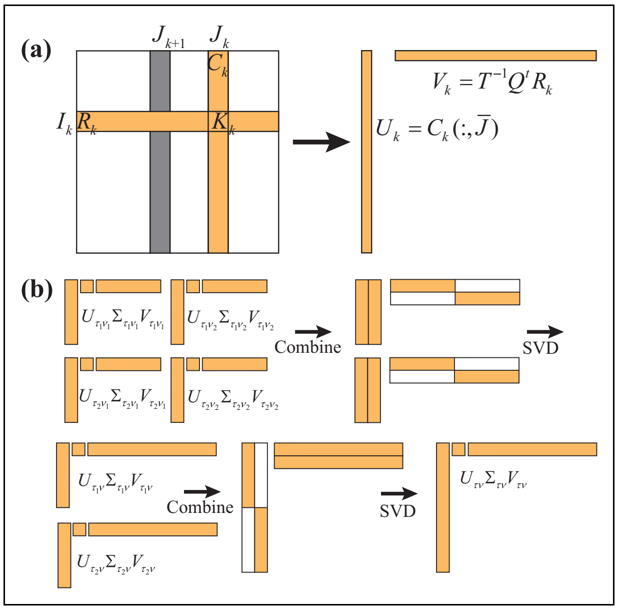
(a)
J_{k+1}
J_k
C_k
I_k
R_k
K_k
$V_k = T^{-1}Q^t R_k$
$U_k = C_k(:, \bar{J})$
(b)
$U_{\tau_1\nu_1}\Sigma_{\tau_1\nu_1}V_{\tau_1\nu_1}$
$U_{\tau_1\nu_2}\Sigma_{\tau_1\nu_2}V_{\tau_1\nu_2}$
$U_{\tau_2\nu_1}\Sigma_{\tau_2\nu_1}V_{\tau_2\nu_1}$
$U_{\tau_2\nu_2}\Sigma_{\tau_2\nu_2}V_{\tau_2\nu_2}$
Combine
SVD
$U_{\tau_1\nu}\Sigma_{\tau_1\nu}V_{\tau_1\nu}$
$U_{\tau_2\nu}\Sigma_{\tau_2\nu}V_{\tau_2\nu}$
Combine
SVD
$U_{\tau\nu}\Sigma_{\tau\nu}V_{\tau\nu}$

Figure 1. (a) Selection of I_k/J_k and form the low-rank update $U_k V_k$. (b) Low-rank merge operation.

$E_{k-1} \approx C_k W_k^\dagger R_k = U_k V_k$ (Voronin and Martinsson, 2017) by (9) and (10). Note that the pseudo inverse is computed via rank-revealing QR (also see the LRID algorithm at line 8). The rank-revealing algorithm is needed as the $d \times d$ block W_k can be further compressed with rank d_k. Particularly for matrices where the ACA algorithm tends to fail, the corresponding $d \times d$ matrices W_k in BACA are often rank-deficient. In this case, BACA becomes more robust than ACA as the effective d_k pivots can still be used to generate d columns J_{k+1} for the next iteration (as long as $d_k > 0$). Consequently, the effective rank increase is $d_k \leq d$ and the pivot pair (I_k, J_k) is updated in (11) by the column pivots $\bar{J}$ of QRCP in (9).

$$[Q, T, \bar{J}] = QR(W_k, \varepsilon) \text{ with } Q \in \mathbb{R}^{d\times d_k} \tag{9}$$

$$U_k = C_k(:, \bar{J}), \quad V_k = T^{-1}Q^t R_k \tag{10}$$

$$I_k \leftarrow I_k([1, d_k]), J_k \leftarrow J_k(\bar{J}) \tag{11}$$

- Compute $\nu = ||U_k V_k||_F$ and update $\mu = ||UV||_F$. Assuming constant block size d, the norm of the low-rank update can be computed in $O(nd_k^2)$ operations (line 11) via

$$T_{U_k} = Chol(U_k^t U_k), T_{V_k} = Chol(V_k V_k^t) \tag{12}$$

$$\nu = ||T_{U_k} T_{V_k}^t||_F \tag{13}$$

Once ν is computed, the norm of UV can be updated efficiently in $O(nr_k d_k)$ operations (line 12) as

$$\mu^2 \leftarrow \mu^2 + \nu^2 + 2\sum_{i=1}^{r_{k-1}}\sum_{j=1}^{d_k} \sim V(i, j)$$

$$\tilde{V} = (VV_k^t) \circ (U^t U_k) \quad (14)$$

where r_k represents the column dimension of U at iteration k. Note that the matrix multiplications in (12) and (14) involving V_k and V (and similarly for those involving U_k and U) can be performed as $[V, V_k]V_k^t$ to further improve the computational efficiency. Then the algorithm updates U, V as $[U, U_k]$, $[V; V_k]$ and tests the stopping criterion $\nu < \varepsilon\mu$. Note that ν, μ with larger d provides better approximations to the exact stop criterion compared to those in (5) hence can significantly reduce the chance of premature termination.

We would like to highlight the difference between the proposed BACA algorithm and existing ACA algorithms. First, as BACA selects a block of rows and columns per iteration as opposed to a single row and column in the baseline ACA algorithm, the convergence behavior and flop performance can be significantly improved. In the existing ACA algorithms, convergence can also be improved by leveraging averaged stopping criteria (Zhou et al., 2017) or searching a single pivot in a broader range of rows and columns (e.g. fully pivoted ACA). However, they still find one row or column at a time in each iteration and hence suffer from poor flop performance. Moreover, they cannot utilize strong rank-revealing algorithms to select skeleton rows and columns with better volume (determinant in modulus) qualities. Second, BACA also has important connections to the hybrid ACA algorithm (Grasedyck and Hackbusch, 2005). The hybrid ACA algorithm assumes prior knowledge about the skeleton rows and columns to leverage interpolation algorithms (e.g. ID and CUR) on a skeleton submatrix and use ACA to refine the skeletons. In contrast, BACA uses cross approximations with QRCP to select skeleton rows and columns and uses interpolation algorithms (LRID at line 8) to form the low-rank update in each iteration. In other words, hybrid ACA can be treated as embedding ACA into interpolation algorithms while BACA can be thought of as embedding interpolation algorithms into ACA iterations. In addition, BACA is purely algebraic and requires no prior knowledge of the row/column skeletons or geometrical information about the rows/columns.

It is worth mentioning that the choice of d affects the trade-off between efficiency and robustness of the BACA algorithm. When $d < r$, the algorithm requires $O(nr^2)$ operations assuming convergence in $O(r/d)$ iterations as each iteration requires $O(nr_k d)$ operations. For example, BACA (Algorithm 2) precisely reduces to ACA (Algorithm 1) when $d = 1$. In what follows we refer to the baseline ACA algorithm as BACA with $d = 1$. On the other hand, BACA converges in a constant number of iterations when $d \gg r$. In the extreme case, BACA reduces to QRCP-based ID when $d = \min\{m, n\}$ (note that the LRID algorithm at line 8 remains the only nontrivial operation). In this case, the algorithm requires $O(n^2 r)$ operations but enjoys the provable convergence of QRCP. Detailed complexity analysis of the BACA algorithm will be provided in Section 4.

The BACA algorithm oftentimes exhibits overestimated ranks compared to those revealed by truncated SVD. Therefore, an SVD re-compression step of U and V may be needed via first computing a QR of U and V as $[Q_U, T_U] = QR(U)$, $[Q_V, T_V] = QR(V^t)$, and then a truncated SVD of $T_U T_V^t$ (Heldring et al., 2015). The result can be viewed as an approximate truncated SVD of A and we assume this is the output of the BACA algorithm in the rest of this article.

3.3. Parallel hierarchical low-rank merge

The distributed-memory implementations of the proposed BACA algorithm and the baseline ACA algorithm can pose performance challenges as straightforward parallelization of all operations in Algorithm 2 and 1 involves many collective communications. To see this, assuming the U and V factors in Algorithm 1 follow 1-D block row and column data layouts, then every operation from line 3 to line 9 requires one or more collective communications. Instead, one can assign one process to perform BACA/ACA on submatrices without any communication and then leverage parallel low-rank arithmetic to merge the results into one single low-rank product. To elucidate the proposed algorithm, we first describe the hierarchical low-rank merge algorithm, then outline its parallel implementation.

Given a matrix $A \in \mathbb{R}^{m\times n}$ with $m \approx n$, the algorithm first creates L-level binary trees for index vectors $[1, m]$ and $[1, n]$ with index set I_τ and J_ν for nodes τ and ν at each level, upon recursively dividing each index set into I_{τ_i}/J_{ν_j} of approximately equal sizes, $i = 1, 2$, $j = 1, 2$. Here, τ_i and ν_j are children of τ and ν, respectively. The leaf and root levels are denoted 0 and L, respectively. This process generates n_b leaf-level submatrices of similar sizes. For simplicity, it is assumed $n_b = 4^L$. We denote submatrices associated with τ, ν as $A_{\tau\nu} = A(I_\tau, J_\nu)$ and their truncated SVD as $[U_{\tau\nu}, \Sigma_{\tau\nu}, V_{\tau\nu}, r_{\tau\nu}] = SVD(A_{\tau\nu}, \varepsilon)$. Here $r_{\tau\nu}$ is the ε-rank of $A_{\tau\nu}$. As submatrices $A_{\tau\nu}$ have significantly smaller dimensions than A (e.g. when $n_b = O(n^2)$ as an extreme case), both BACA and ACA algorithms become more robust to attain the truncated SVD. Following compression of n_b submatrices $A_{\tau\nu}$ by BACA or ACA at step $l = 0$, there are multiple approaches to combine them into one low-rank product including randomized algorithms via applying A to random matrices, and deterministic algorithms via recursively pair-wise re-compressing the blocks using low-rank arithmetic. Here we choose the deterministic algorithm for simplicity of rank estimation and parallelization. Here, we deploy truncated SVD as the re-compression tool but other tools such as ID, QR, UTV can also be applied. Figure 1(b) illustrates one re-compression operation for transforming SVDs of $A_{\tau_i\nu_j}, i = 1, 2, j = 1, 2$ into that of $A_{\tau\nu}$. The operation first horizontally compresses SVDs of $A_{\tau_i\nu_j}, i = 1, 2, j = 1, 2$ at step $l - \frac{1}{2}$ and then vertically compresses the results, that is, SVDs of $A_{\tau_i\nu}, i = 1, 2$ at step l, $l = 1, .., L$. Specifically, the horizontal compression step is composed of one

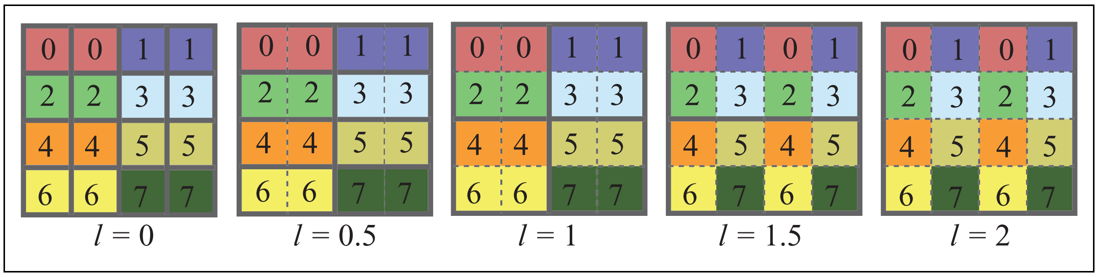

Figure 2. Parallel hierarchical merge with eight processes. Blocks surrounded by solid lines represent $A_{\tau\nu}$ after compression at each step l. Blocks surrounded by dashed lines represent ScaLAPACK blocks.

concatenation operation in (15) and one compression operation in (16):

$$\bar{U}_{\tau_i\nu} = [U_{\tau_i\nu_1}\Sigma_{\tau_i\nu_1}, U_{\tau_i\nu_2}\Sigma_{\tau_i\nu_2}],\ \bar{V}_{\tau_i\nu} = diag(V_{\tau_i\nu_1}, V_{\tau_i\nu_2}) \quad (15)$$

$$[U_{\tau_i\nu},\Sigma_{\tau_i\nu},V_{\tau_i\nu},r_{\tau_i\nu}] \leftarrow SVD(\bar{U}\tau_i\nu,\varepsilon),\ V_{\tau_i\nu} \leftarrow V_{\tau_i\nu}\bar{V}_{\tau_i\nu} \quad (16)$$

with $i = 1, 2$. Let $\bar{U}_{\tau_i\nu}\bar{V}_{\tau_i\nu}$ and $U_{\tau_i\nu}\Sigma_{\tau_i\nu}V_{\tau_i\nu}$ denote the submatrix before and after the SVD truncation, respectively. Similarly, the vertical compression step can be performed via horizontal merge of $A^t_{\tau_i\nu}, i = 1, 2$. Let s_l represent the maximum rank $r_{\tau\nu}$ among all blocks at steps $l = 0, 1, \ldots, L$. Note that the algorithm returns an approximate truncated SVD after L steps. As an example, the hierarchical merge algorithm with the level count of the hierarchical merge $L = 2$ and $n_b = 16$ is illustrated in Figure 2. At step $l = 0$, the algorithm compresses all n_b submatrices with BACA; at step $l = 0.5, 1.5$, the algorithm merges every horizontal pair of blocks; similarly at level $l = 1, 2$, the algorithm merges every vertical pair of blocks. Note that blocks surrounded by solid lines represent results after compression at each step l.

The above-described hierarchical algorithm with BACA for leaf-level compressions is dubbed H-BACA (Algorithm 3). In the following, a distributed-memory implementation of the H-BACA algorithm is described. Without loss of generality, it is assumed that $m = n = 2^i$ and $p = 2^j$. The proposed parallel implementation first creates two $\log\sqrt{p}$-level binary trees with p denoting the total number of MPI processes. One process performs BACA compression of one or two leaf-level submatrices and low-rank merge operations from the bottom up until it reaches a submatrix shared by more than one process. Then, all such blocks are handled by PBLAS and ScaLAPACK with BLACS process grids that aggregate those in corresponding submatrices. Consider the example in Figure 2 with process count $p = 8$. The workload of each process is labeled with its process rank and highlighted with one color. The dashed lines represent the ScaLAPACK blocks. First, BACA compressions and merge operations at $l = 0, 0.5$ are handled locally by one process without any communication. Next, merge operations at $l = 1, 1.5, 2$ are handled by BLACS grids of 2×1, 2×2, and 4×2, respectively. For illustration purposes, we select the ScaLAPACK block size in Figure 2 as $n_0 \times n_0$ where n_0 is the dimension of the finest-level submatrices in the hierarchical merge algorithm and $n = \sqrt{n_b}n_0$. In this case, the only required data redistribution is from step $l = 1$ to $l = 1.5$. However, the ScaLAPACK block size may be set to much smaller numbers in practice, requiring data redistribution at each row/column re-compression step. Similarly, the requirement of $m = n = 2^i$ and $p = 2^j$ is not needed in practice.

Algorithm 3. Hierarchical low-rank merge algorithm with BACA (H-BACA).

```
input : Matrix $A \in \mathbb{R}^{m\times n}$, number of leaf-level
        subblocks $n_b$, block size $d$ of leaf-level
        BACA, relative tolerance $\epsilon$
output: Truncated SVD of $A \approx U\Sigma V$ with rank $r$
1  Create $L$-level trees on index vectors $[1, m]$ and
     $[1, n]$ with index set $I_\tau$ and $J_\nu$ for nodes $\tau$ and $\nu$
     at each level, $L = \log\sqrt{n_b}$, the leaf and root levels
     are denoted 0 and $L$, respectively;
2  for $l = 0$ to $L$ do
3      foreach $A_{\tau\nu} = A(I_\tau, J_\nu)$ at level $l$ do
4          if leaf-level then
5              $[U_{\tau\nu}, \Sigma_{\tau\nu}, V_{\tau\nu}, r_{\tau\nu}] =$
                 $BACA(A_{\tau\nu}, d, \epsilon)$;
6          else
7              Let $\tau_1, \tau_2$ and $\nu_1, \nu_2$ denote children of
                 $\tau$ and $\nu$;
8              for $i = 1$ to 2 do
9                  $\bar{U}_{\tau_i\nu} = [U_{\tau_i\nu_1}\Sigma_{\tau_i\nu_1}, U_{\tau_i\nu_2}\Sigma_{\tau_i\nu_2}]$;
10                 $\bar{V}_{\tau_i\nu} = \mathrm{diag}(V_{\tau_i\nu_1}, V_{\tau_i\nu_2})$;
11                 $[U_{\tau_i\nu}, \Sigma_{\tau_i\nu}, V_{\tau_i\nu}, r_{\tau_i\nu}] \leftarrow$
                     $SVD(\bar{U}_{\tau_i\nu}, \epsilon)$;
12                 $V_{\tau_i\nu} \leftarrow V_{\tau_i\nu}\bar{V}_{\tau_i\nu}$;
13             $\bar{U}_{\tau\nu} = diag(U_{\tau_1\nu}, U_{\tau_2\nu})$;
14             $\bar{V}_{\tau\nu} = [\Sigma_{\tau_1\nu}V_{\tau_1\nu}; \Sigma_{\tau_2\nu}V_{\tau_2\nu}]$;
15             $[U_{\tau\nu}, \Sigma_{\tau\nu}, V_{\tau\nu}, r_{\tau\nu}] \leftarrow SVD(\bar{V}_{\tau\nu}, \epsilon)$;
16             $U_{\tau\nu} \leftarrow \bar{U}_{\tau_\nu}U_{\tau\nu}$;
17 return $U = U_{\tau\nu}, V = V_{\tau\nu}, \Sigma = \Sigma_{\tau\nu}, r = r_{\tau\nu}$;
```

4. Cost analysis

In this section, the costs for computation and communication of the proposed BACA and H-BACA algorithms are analyzed.

4.1. Computational cost

First, the costs for BACA can be summarized as follows. Assuming BACA converges in $O(r/d)$ iterations, each iteration performs entry evaluation from the residual matrices, QRCP for pivot selection, LRID for forming the LR product, and estimation of matrix norms. The entry evaluation computes $O(nd)$ entries each requiring $O(r_k)$ operations; QRCP on block rows requires $O(nd^2)$ operations; the LRID algorithm requires $O(ndd_k + d_kd^2)$ operations; norm estimation requires $O(nr_kd_k)$ operations. Summing up these costs, the overall cost for the BACA algorithm is

$$c_{BACA} = \sum_{k=1}^{O(r/d)} (nd^2 + nr_kd + d_kd^2)$$

$$\leq O(nd^2 + rd^2 + nrd)O(r/d) = O(nr^2) \quad (17)$$

Here we assume the block size $d \leq r$. Note that when $d \gg r$ (e.g. $d = O(n)$), it follows that the worst-case complexity is $c_{BACA} = O(n^2r)$ by bypassing the pivot selection step that causes the nd^2 term. In practice, one would always avoid the case of $d \gg r$.

Next, the computational costs of the H-BACA algorithm are analyzed. The costs are analyzed for two cases of distributions of the maximum ranks s_l at each level, that is, $s_l = r$ (ranks stay constant during the merge) and $s_l \approx 2^l r/\sqrt{n_b} = 2^{l-L}r$ (rank increases by a factor of 2 per level), $l = 0, 1, \ldots, L$. The constant rank case is often valid for matrices with their numerical ranks independent of matrix dimensions (e.g. random low-rank matrices, matrices representing well-separated interactions from low-frequency and static wave equations and certain quantum chemistry matrices); the increasing-rank case holds true for matrices whose ranks depend polynomially (with order no bigger than 1) on the matrix dimensions (e.g. those arising from high-frequency wave equations, matrices representing near-field interactions from low-frequency and static wave equations, and certain classes of kernel methods on high dimensional data sets). Note that these rank distributions often follow from a proper hierarchical partitioning tree and may not be valid using an arbitrary partitioning tree. From the aforementioned analysis of BACA, the computational costs for the leaf-level compression $c_b = c_{BACA}n_b$ are:

$$c_b = O(\frac{n}{\sqrt{n_b}} s_0^2 n_b), \quad \text{if } d \leq s_0 \quad (18)$$

which represent the complexity with ACA when $n_b = 1$.

Let $n_l = 2^l n/\sqrt{n_b}$ denote the size of submatrices $A_{\tau,\nu}$ at level l. The computational costs c_m of hierarchical merge operations can be estimated as

$$c_m = \sum_{l=1}^{L} O(4^{L-l} n_l s_l^2) \quad (19)$$

Accounting for the two cases of rank distributions, the computational costs for the leaf-level BACA and hierarchical merge operations of the H-BACA algorithm are summarized in Table 1. Note that the costs of the BACA algorithm can also be extracted from Table 1 upon setting $n_b = 1$. Not surprisingly, the hierarchical merge algorithm induces a computational overhead of at most $\sqrt{n_b}$ when ranks stay constant; the leaf-level compression can have a $1/\sqrt{n_b}$ reduction factor for the increasing rank case and $\sqrt{n_b}$ overhead for the constant rank case.

Table 1. Flop counts and communication costs for the leaf-level compression and hierarchical merge operations in Algorithm 3 for two classes of low-rank matrices.[a]

	Constant rank $s_l \approx r$	Increasing rank $s_l \approx r/\sqrt{n_b} \times 2^l$
BACA $d \leq s_0$	$O(nr^2\sqrt{n_b})$	$O(nr^2)/\sqrt{n_b}$
Merge compute	$O(nr^2\sqrt{n_b})$	$O(nr^2)$
Merge communicate	$[O(r\log^2 p), O(nr\log^2 p/\sqrt{p})]$	$[O(r\log p), O(nr\log p/\sqrt{p})]$

[a]n and r denote matrix dimension and rank. d denotes the block size in BACA. p and n_b denote number of processes and leaf-level submatrices. s_l denotes maximum ranks among all level-l submatrices.

For completeness, the comparison between the proposed BACA, H-BACA algorithms (assuming $d \leq r_0$) and existing ACA algorithms are given in Table 2. In contrast to existing ACA algorithms that select one pivot at a time, BACA and H-BACA select d and n_bd pivots simultaneously. As such, H-BACA is the most robust algorithm among all listed here. Not surprisingly, H-BACA can induce a computational overhead of $\sqrt{n_b}$.

4.2. Communication cost

As the leaf-level BACA compression requires no communication, only the communication costs for the hierarchical merge operations are analyzed here. Since the merge operations may introduce an $O(\sqrt{n_b})$ computational overhead, one would only increase n_b to create more parallelism, i.e. the process count $p \approx n_b$. Consider the parallelization of one level l merge operation, that is, transformation of the SVDs of $A_{\tau_i\nu_j}, i = 1,2, j = 1,2$ into that of $A_{\tau\nu}$ shown in Figure 1(b). Let $p_l = 4^l$ denote the number of processes sharing one level l block $A_{\tau\nu}$, $l = 0, \ldots, L$. The horizontal compression step in (15, 16) requires redistribution from the process grids of size p_{l-1} sharing $A_{\tau_i\nu_j}, i = 1,2, j = 1,2$ to the process grids of size $2p_{l-1}$ sharing $A_{\tau_i\nu}, i = 1,2$. After redistribution, each process grid involves a PDGESVD function in ScaLAPACK (see (16)) to compute the new rank after the combination in (15), and a PDGEMM function in PBLAS to multiply factors $V_{\tau_i\nu}$ with $\bar{V}_{\tau_i\nu}$ (see 16). Similarly, the vertical compression step requires redistribution from the process grids of size $2p_{l-1}$ sharing $A_{\tau_i\nu}, i = 1,2$ to the process grids of size p_l sharing $A_{\tau\nu}$, and calling PDGESVD and PDGEMM functions in the new grids. Let the pair [#messages, volume] denote the communication cost including the number of messages and the number of words transferred along the critical path. Then the communication costs for each

Table 2. Comparisons between proposed BACA, H-BACA algorithms and existing ACA algorithms.[a]

Algorithm	ACA/ACA$^+$	Hybrid-ACA	BACA	H-BACA
Pivot count per iteration	1	1	d	$n_b d$
Cost (constant rank)	$O(nr^2)$	$O(nr^2)$	$O(nr^2)$	$O(nr^2\sqrt{n_b})$
Cost (increasing rank)	$O(nr^2)$	$O(nr^2)$	$O(nr^2)$	$O(nr^2)$
Preselection of submatrices	No	Yes	No	No

[a]Note that the algorithms show increasing robustness from left to right.

(BLACS) grid redistribution, PDGEMM and PDGESVD during the hierarchical merge are $[O(1), O(n_l s_l/p_l)]$, $[O(s_l), O(n_l s_l/\sqrt{p_l})]$, and $[O(s_l \log p_l), O(n_l s_l \log p_l/\sqrt{p_l})]$, respectively (Blackford et al., 1997). Recall that $n_l = 2^l n/\sqrt{p}$ and s_l denote the size and rank of submatrices at level l and note that $n_l \gg s_l$. Therefore the communication cost v_m of the hierarchical merge (and H-BACA) can be estimated as

$$v_m = \sum_{l=1}^{L}\left[O(s_l \log p_l), O\left(\frac{n_l s_l \log p_l}{\sqrt{p_l}}\right)\right] = \sum_{l=1}^{L}\left[O(l s_l), O\left(\frac{l n s_l}{\sqrt{p}}\right)\right] \tag{20}$$

Consider the two cases of rank distributions, that is, $s_l = r$ and $s_l \approx 2^{l-L} r$, the overall communication costs of H-BACA are $v_m = [O(r\log^2 p), O(nr\log^2 p/\sqrt{p})]$ and $v_m = [O(r\log p), O(nr\log p/\sqrt{p})]$, respectively (see Table 1).

5. Numerical results

This section presents several numerical results to demonstrate the accuracy and efficiency of the proposed H-BACA algorithm. The matrices in all numerical examples are generated from the following kernels:

1. Gaussian kernel: $A_{i,j} = \exp(\frac{-||x_i - x_j||^2}{2h^2})$, $i,j = 1, \ldots, 2n$. Here h is the Gaussian width, and $x_i \in \mathbb{R}^{8\times 1}$ and $\mathbb{R}^{784\times 1}$ are feature vectors in one subset of the SUSY and MNIST Data Sets from the UCI Machine Learning Repository (Dheeru and Karra Taniskidou, 2017), respectively. Note that the Gaussian kernel permits low-rank compression as shown in (Bach, 2013; Musco and Musco, 2017; Wang et al., 2018).
2. EFIE2D kernel: $A_{i,j} = H_0^{(2)}(k||x_i - x_j||)$ resulting from the Nyström discretization of the electric field integral equation (EFIE) for electromagnetic scattering from 2-D curves. Here $H_0^{(2)}$ is the second kind Hankel function of order 0, k is the free-space wavenumber, $x_i, x_j \in \mathbb{R}^{2\times 1}$ are discretization points (15 points per wavelength) of two 2-D parallel strips of length 1 and distance 1.
3. EFIE3D kernel: A is obtained by the Galerkin method for EFIE to analyze electromagnetic scattering from 3-D surfaces.
4. Frontal3D kernel: A is a dense frontal matrix that arises from the multifrontal sparse elimination for the finite-difference frequency-domain solution of the homogeneous-coefficient Helmholtz equation inside a unit cube.
5. Polynomial kernel: $A_{i,j} = (x_i^t x_j + h)^2$. Here $x_i, x_j \in \mathbb{R}^{50\times 1}$ are points from a randomly generated data set, and h is a regularization parameter.
6. Product-of-random kernel: $A = UV$ with $U \in \mathbb{R}^{n\times r}$ and $V \in \mathbb{R}^{r\times n}$ being random matrices with i.i.d. entries.

Note that the EFIE2D, EFIE3D, and Frontal3D kernels result in complex-valued matrices. Throughout this section, we refer to ACA as a special case of BACA when $d = 1$. In all examples except for the product-of-random kernel, the algorithm is applied to the offdiagonal submatrix $A_{12} = A(1:n, 1+n:2n)$ assuming rows/columns of A have been properly permuted (e.g. by a KD-tree partitioning scheme). Note that the permutation may yield a hierarchical matrix representation of A, but in this article we only use the permutation to define the partition trees for H-BACA compression of one off-diagonal subblock of A with H-BACA. All experiments are performed on the Cori Haswell machine at NERSC, which is a Cray XC40 system and consists of 2388 dual-socket nodes with Intel Xeon E5-2698v3 processors running 16 cores per socket. The nodes are configured with 128 GB of DDR4 memory at 2133 MHz.

5.1. Convergence

First, the convergence of the proposed BACA algorithm is investigated using several matrices: Gaussian-SUSY matrices with $n = 5000$, $h = 1.0, 0.2$, an EFIE3D matrix for a unit sphere with $n = 21788$ and approximately 20 points per wavelength, and a Frontal3D matrix with $n = 1250$ and 10 points per wavelength. The corresponding ε-ranks are $r = 4683, 1723, 1488, 718$ for $\varepsilon = 10^{-6}$. The residual histories versus revealed ranks r_k, at each iteration k of BACA with $1 \leq d \leq 256$ are plotted in Figure 3. The residual error is defined as $||U_k V_k||_F/||UV||_F$ from (13). As a reference, the singular value spectra $\Sigma(k,k)/\Sigma(1,1)$ computed from $[U, \Sigma, V, r] = SVD(A, \varepsilon)$ are also plotted.

For the Gaussian-SUSY matrices, the baseline ACA algorithm ($d = 1$) behaves poorly with smaller h due to the exponential decay of the Gaussian kernel. As a result, the matrix becomes increasingly sparse and coherent for small h particularly for high dimensional data sets. In fact, ACA

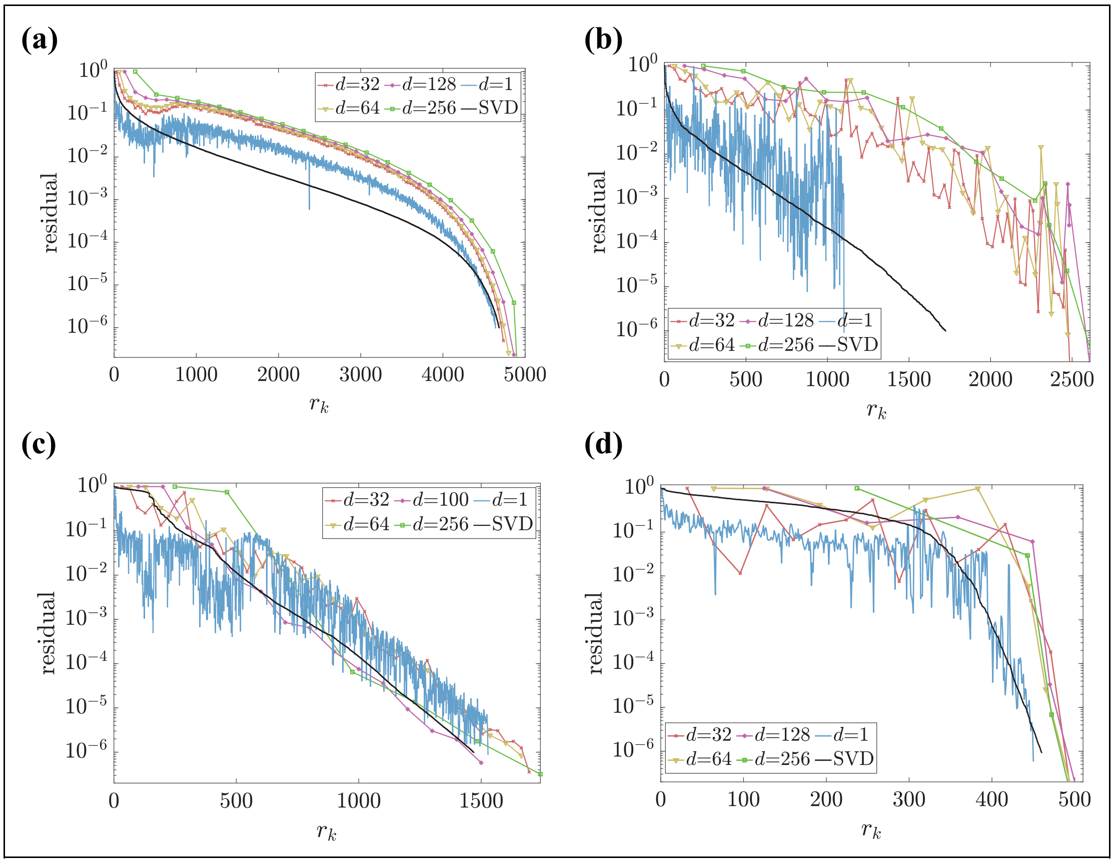

Figure 3. Convergence history of BACA for the (a) Gaussian-SUSY kernel with $h = 1.0$, $n = 5000$, $\varepsilon = 10^{-6}$, $r = 4683$, (b) Gaussian-SUSY kernel with $h = 0.2$, $n = 5000$, $\varepsilon = 10^{-6}$, $r = 1723$, (c) EFIE3D kernel for a unit sphere with $n = 21,788$, $\varepsilon = 10^{-6}$, $r = 1488$, and (d) Frontal3D kernel with $n = 1250$, $\varepsilon = 10^{-6}$, $r = 718$.

constantly selects smaller pivots and the residual exhibits wild oscillations particularly for smaller h (e.g. when $h = 0.2$ in Figure 3(b)). Similarly, the analytical and numerical Green's functions respectively for the EFIE3D (Figure 3(c)) and Frontal3D (Figure 3(d)) matrices are not asymptotically smooth for ACA to converge rapidly. For all examples in Figure 3, significant portions of the residual curves lie below the singular value spectra which causes premature iteration termination for certain given residual errors. In stark contrast, the proposed BACA algorithm ($d = 32, 64, 100, 128, 256$) shows increasingly smooth residual histories residing above the singular value spectra as the block size d increases. Although BACA may overestimate the matrix ranks particularly for larger d, the SVD re-compression step mentioned in Section 3.2 can effectively reduce the ranks.

5.2. Accuracy

Next, the accuracy of the H-BACA algorithm is demonstrated using the following matrices: two Gaussian-SUSY matrices with $n = 5000$, $h = 1.0, 0.2$, one EFIE3D matrix for a unit sphere with $n = 1707$ and approximately 20 points per wavelength, and a Frontal3D matrix with $n = 1250$ and 10 points per wavelength. The relative Frobenius-norm error $||A - UV||_F/||A||_F$ is computed for changing number of leaf-level submatrices n_b and block size d. When $h = 1.0$ for the Gaussian-SUSY matrix (Figure 4(a)), the H-BACA algorithms achieve desired accuracies ($\varepsilon = 10^{-2}, 10^{-6}, 10^{-10}$) using the baseline ACA ($d = 1$), and BACA ($d = 32$) when $n_b = 1$ and the hierarchical merge operation only causes slight error increases as n_b increases. However when $h = 0.2$ for the Gaussian-SUSY matrix (Figure 4(b)), all data points for H-BACA with $d = 1$ fail due to the wildly oscillating residual histories. In contrast, H-BACA with $d = 32$ achieves significantly better accuracies for most data points particularly as n_b increases. For the EFIE3D (Figure 4(c)) and Frontal3D (Figure 4(d)) matrices, H-BACA with $d = 32$ achieves comparable accuracies as H-BACA with $d = 1$ for most data points. Note that $d = 32$ is significantly better than $d = 1$ when the prescribed residual error is large ($\varepsilon = 10^{-2}$). This agrees with the residual histories in Figure 3(c) and (d) as they lie below the singular value spectra when iteration count k is small.

5.3. Efficiency

This subsection provides six examples to verify the computational complexity estimates in Table 1. H-BACA

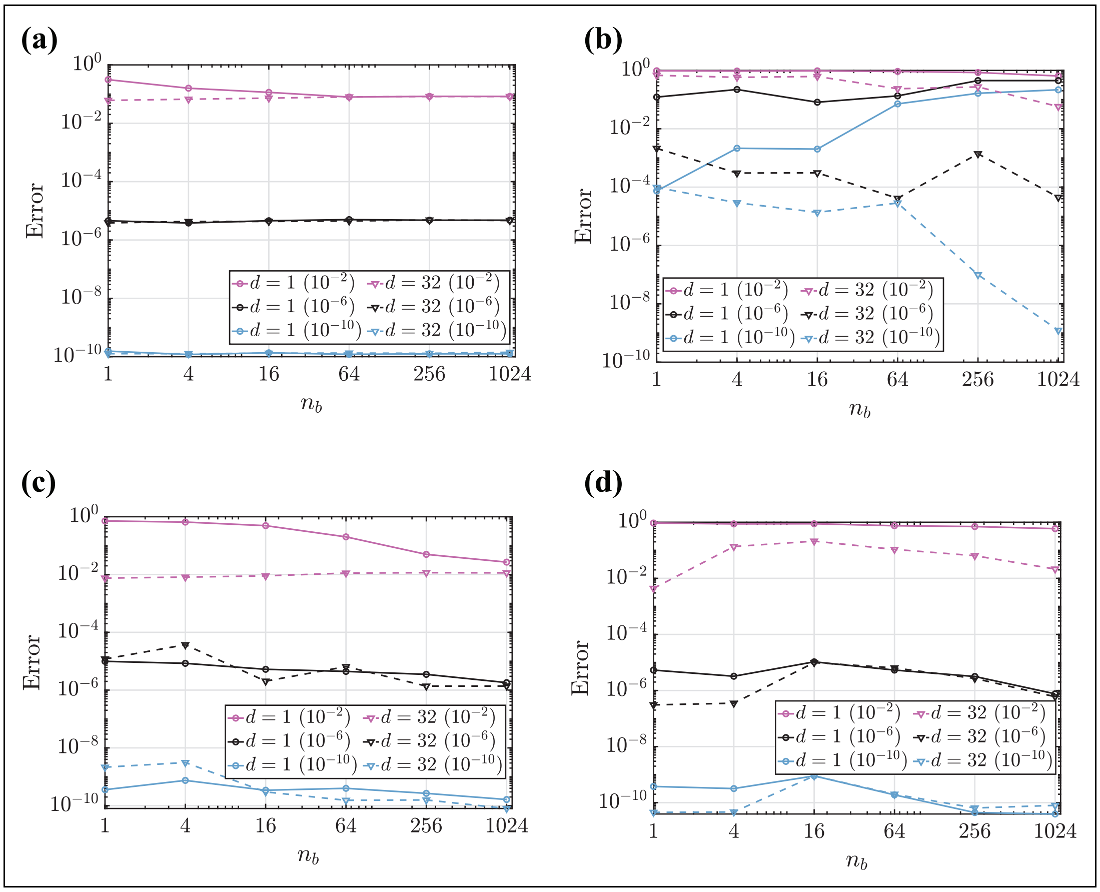

Figure 4. Measured error of H-BACA with $\varepsilon = 10^{-2}, 10^{-6}, 10^{-10}$ for the (a) Gaussian-SUSY kernel with $h = 1.0$, $n = 5000$, (b) Gaussian-SUSY kernel with $h = 0.2$, $n = 5000$, (c) EFIE3D kernel for a unit sphere with $n = 1707$, and (d) Frontal3D kernel with $n = 1250$.

with leaf-level ACA ($d = 1$) and BACA ($d = 8, 16, 32, 64, 128$) is tested for the following matrices: one Gaussian-SUSY matrix with $n = 50,000$, $h = 1.0$, $\varepsilon = 10^{-2}$, one Gaussian-MNIST matrix with $n = 5000$, $h = 3.0$, $\varepsilon = 10^{-2}$, one EFIE3D matrix for a unit sphere with $n = 26268$, $\varepsilon = 10^{-6}$ and 20 points per wavelength, one Frontal3D matrix with $n = 1250$, $\varepsilon = 10^{-6}$ and 10 points per wavelength, one polynomial matrix with $n = 10,000$, $h = 0.2$, $\varepsilon = 10^{-4}$, and one product-of-random matrix with $n = 2500$, $\varepsilon = 10^{-4}$. The corresponding ε-ranks are 298, 137, 1488, 788, 450, and 1000, respectively. It can be validated that the hierarchical merge operation attains increasing ranks for the Gaussian, EFIE3D and Frontal3D matrices, and relatively constant ranks for the polynomial, and product-of-random matrices. All examples use one process except that the Gaussian-SUSY example uses 16 processes. The CPU times are measured and plotted in Figure 5.

Table 1 predicts that H-BACA exhibits increasing (with a factor of $\sqrt{n_b}$) and constant time when s_l stays constant and increases, respectively. Note that the rank assumption $s_l \approx r$ leading to the $O(\sqrt{n_b})$ computational overhead may not be fully observed for practical values of n_b and n. Given one matrix, s_l may stay approximately constant for a limited number of subdivision levels l. For example, s_l stay constant for bottom levels of EFIE3D and Frontal3D matrices, and top levels of Polynomial and product-of-random matrices. This agrees with the observed scalings (w.r.t n_b) in Figure 5(c) to (f). As a reference, the $O(\sqrt{n_b})$ curves are plotted and only small ranges of n_b exhibit the $O(\sqrt{n_b})$ overhead. For the Gaussian matrices, we even observe nonincreasing CPU time w.r.t. n_b when n_b is not too big (see Figure 5(a) and (b)).

The effects of varying block size d also deserve further discussions. First, larger block size d can significantly improve the robustness of H-BACA for the Gaussian matrices. For example, H-BACA does not achieve desired accuracies due to premature termination for all data points on the $d = 1$ curve in Figure 5(a) and $d = 1$ and $d = 8$ curves in Figure 5(b). In contrast, H-BACA with larger d attains desired accuracies. Second, larger block size d results in reduced CPU time for the Polynomial and Frontal3D matrices due to better BLAS performance (see Figure 5(d) and (e)). For the other tested matrices, no significant performance differences have been observed by changing block size d. However, for matrices with ranks $s_0 \leq d$, larger d and n_b can introduce significant overheads.

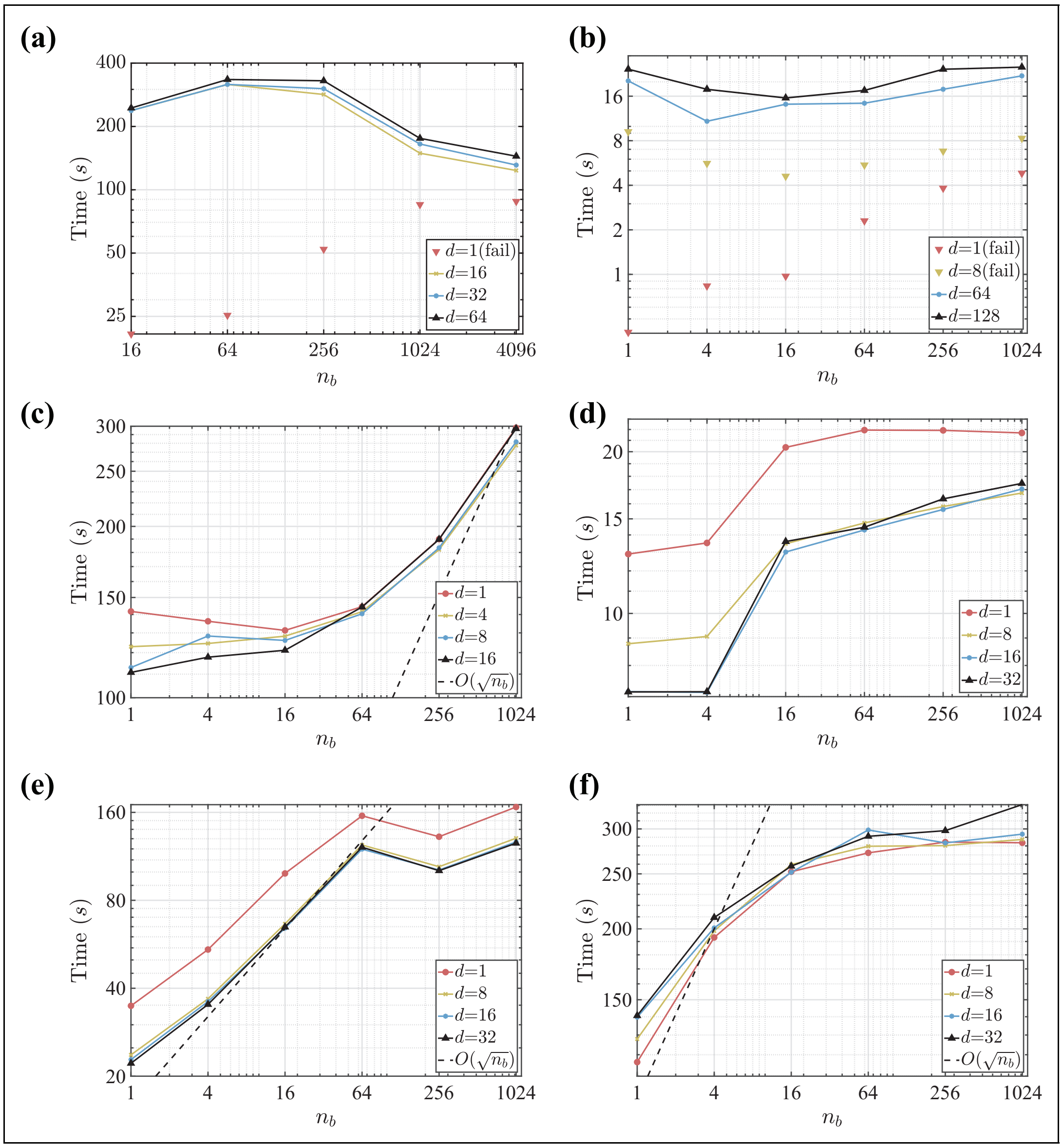

Figure 5. Computation time of H-BACA with varying n_b and d for the (a) Gaussian-SUSY kernel with $h = 1.0$, $n = 50,000$, $\varepsilon = 10^{-2}$, $r = 298$, (b) Gaussian-MNIST kernel with $h = 3.0$, $n = 5000$, $\varepsilon = 10^{-2}$, $r = 137$, (c) EFIE3D kernel for a unit sphere with $n = 26,268$, $\varepsilon = 10^{-6}$, $r = 1488$, (d) Frontal3D kernel with $n = 1250$, $\varepsilon = 10^{-6}$, $r = 788$, (e) polynomial kernel with $h = 0.2$, $n = 10,000$, $\varepsilon = 10^{-4}$, $r = 450$, and (f) product-of-random kernel with $n = 2500$, $r = 1000$. Note that the data points where the algorithm fails are shown as triangular markers without lines.

5.4. Parallel performance

Finally, the parallel performance of the H-BACA algorithm is demonstrated via strong scaling studies with the EFIE2D, EFIE3D, product-of-random and Gaussian matrices with process counts $p = 8, \ldots, 1024$. For the EFIE2D matrices, $n = 160,000$ and the wavenumbers are chosen such that the ε-ranks with $\varepsilon = 10^{-4}$ are 937 and 107, respectively. For the EFIE3D matrices for a unit square, $n = 21,788$ and the wavenumbers are chosen such that the ε-ranks with $\varepsilon = 10^{-6}$ are 1007 and 598, respectively. For the product-of-random matrices, $n = 10,000$ and the inner dimension of the product is set to $r = 2000$ and 800, respectively. For the Gaussian matrices with a randomly generated data set of dimension 50 and $n = 10,000$, we choose $h = 1.0$ and $h = 1.6$ such that the ε-ranks with $\varepsilon = 10^{-3}$ are 2106 and 191, respectively. In all examples, the block size and number of leaf-level subblocks in H-BACA are chosen as $d = 8$ and $\sqrt{n_b} = \sqrt{p}$. The ScaLAPACK block size is set to 64×64. As the reference, we compare to a straightforward parallel implementation of the baseline ACA algorithm which essentially parallelize every operation in ACA with collective MPI communications.

For all examples, the parallel ACA algorithm stops scaling when p is sufficiently large (see Figure 6). In contrast, the proposed parallel H-BACA algorithm scales up to $p = 1024$. In most examples, H-BACA achieves better

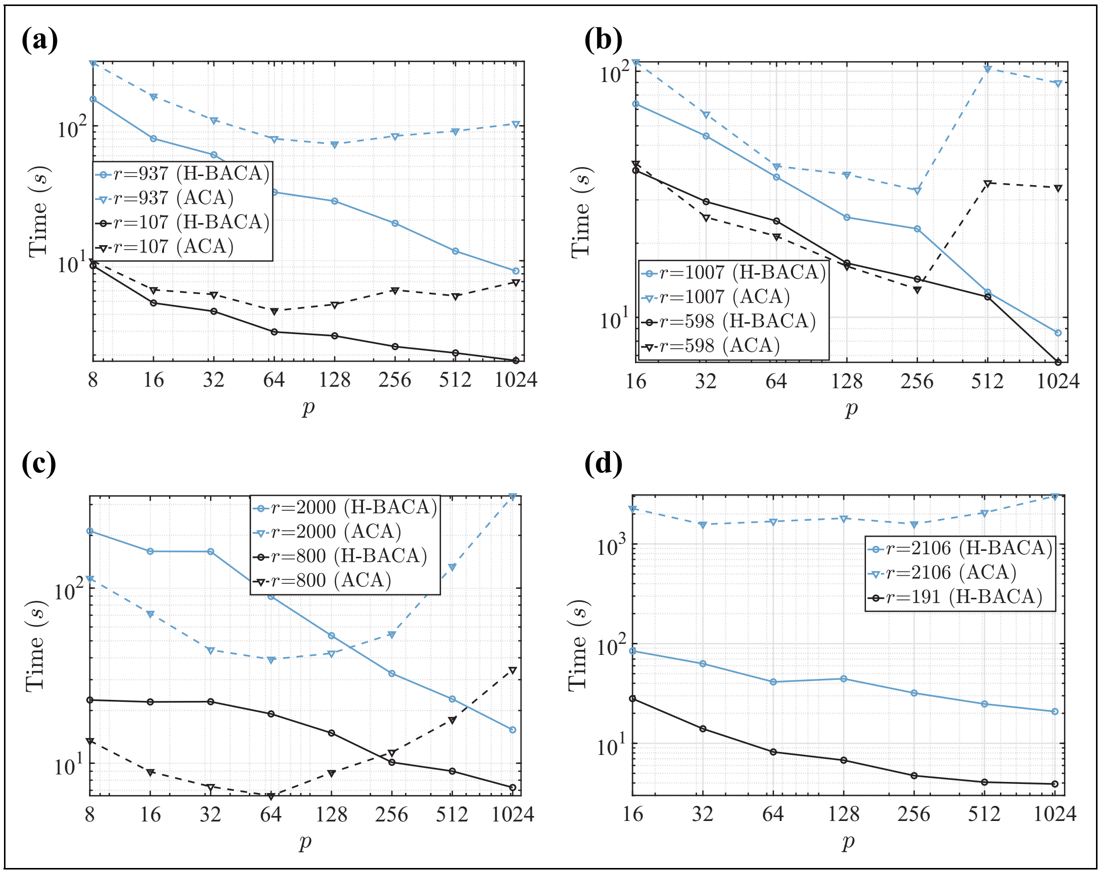

Figure 6. Computation time of H-BACA with varying process counts for the (a) EFIE2D kernel with $n = 160,000$, $\varepsilon = 10^{-4}$, $r = 107$ and 937, (b) EFIE3D kernel for a unit square with $n = 21,788$, $\varepsilon = 10^{-6}$, $r = 598$ and 1007, (c) product-of-random kernel with $n = 10,000$, $r = 800$ and 2000, and (d) Gaussian kernel for a randomly generated data set with $h = 1.0, 1.6$, $\varepsilon = 10^{-3}$, $r = 2106$ and 191. Note that for the Gaussian matrix with $r = 191$, ACA fails to provide accurate results and is not plotted.

parallel efficiencies with larger ranks due to better process utilization during the hierarchical merge operation. We also note that ACA outperforms H-BACA for the Product-of-random matrices with small process count p (and n_b). This is partially attributed to the $O(\sqrt{n_b})$ overhead observed in Figure 5(f).

Overall, the parallel H-BACA algorithm can achieve reasonably good parallel performances for rank-deficient matrices with modest to large numerical ranks. Not surprisingly, the parallel runtime is dominated by that of ScaLAPACK computation and possible redistributions between each re-compression step as analyzed in Section 4. Also note that the leaf-level BACA compression is embarrassingly parallel for all test cases.

6. Conclusion

This article presents a parallel and algebraic ACA-type matrix decomposition algorithm given that any matrix entry can be evaluated in $O(1)$ time. Two proposed strategies, BACA and H-BACA, are leveraged to improve the robustness and parallel efficiency of the (baseline) ACA algorithm for general rank-deficient matrices.

First, the BACA algorithm searches for blocks of row/column pivots via QRCP on the column/row submatrices at each iteration. The blocking nature of BACA provides a closer estimation of the true residual error and reduces the chance of selecting smaller pivots when compared to ACA. Therefore, BACA exhibits a much smoother and more reliable convergence history. Moreover, blocked operations also benefit from higher flop performance compared to non-blocked ones. For a rank-deficient matrix with dimension n and ε-rank r, the computational cost of BACA is $O(nr^2)$ assuming the block size constant and iteration count $O(r)$.

Second, the H-BACA algorithm divides the matrix into n_b similar-sized submatrices each compressed with BACA and then hierarchically merges the results using low-rank arithmetic. Depending on the rank behaviors of submatrices during the merge, the H-BACA may have a computational overhead of $O(\sqrt{n_b})$ yielding the overall computational cost at most $O(nr^2\sqrt{n_b})$. The H-BACA algorithm can be parallelized with distributed-memory machines by assigning each process to one submatrix and leveraging PBLAS and ScaLAPACK for the hierarchical merge operation. Such parallelization strategy yields a much more favorable communication cost when compared to the straightforward parallelization of ACA/BACA with collective MPI routines. Not surprisingly, good parallel performance can be achieved for matrices with modest to large numerical ranks which increases process utilization for each merge operation.

In contrast to the baseline ACA algorithm, the proposed algorithms exhibit improved robustness and favorable parallel performance with low computational overheads for broad ranges of matrices arising from many science and engineering applications.

Authors' note

This research used resources of the National Energy Research Scientific Computing Center (NERSC), a US Department of Energy Office of Science User Facility operated under contract no. DE-AC02-05CH11231.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the US Department of Energy Office of Science and the National Nuclear Security Administration, and in part by the US Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Scientific Discovery through Advanced Computing (SciDAC) program through the FASTMath Institute under contract no. DE-AC02-05CH11231 at Lawrence Berkeley National Laboratory.

ORCID iD

Yang Liu https://orcid.org/0000-0003-3750-1178

References

Bach F (2013) Sharp analysis of low-rank kernel matrix approximations. In: *Proceedings of the 26th annual conference on learning theory, Proceedings of machine learning research*, vol. 30. Princeton, USA: PMLR, 12–14 June 2013, pp. 185–209.

Balzano L, Nowak R and Recht B (2010) Online identification and tracking of subspaces from highly incomplete information. In: *2010 48th annual Allerton conference on communication, control, and computing (Allerton)*, pp. 704–711. DOI: 10.1109/ALLERTON.2010.5706976.

Bebendorf M (2000) Approximation of boundary element matrices. *Numerische Mathematik* 86(4): 565–589.

Bebendorf M and Grzhibovskis R (2006) Accelerating Galerkin BEM for linear elasticity using adaptive cross approximation. *Mathematical Methods in the Applied Sciences* 29(14): 1721–1747.

Blackford LS, Choi J, Cleary A, et al. (1997) *ScaLAPACK Users' Guide*. Philadelphia: Society for Industrial and Applied Mathematics.

Boutsidis C, Mahoney MW and Drineas P (2009) An improved approximation algorithm for the column subset selection problem. In: *Proceedings of the twentieth annual ACM-SIAM symposium on discrete algorithms, SODA '09*, Philadelphia: Society for Industrial and Applied Mathematics, New York, New York, USA, 4–6 January 2009, pp. 968–977.

Candès EJ and Recht B (2009) Exact matrix completion via convex optimization. *Foundations of Computational Mathematics* 9(6): 717.

Cheng H, Gimbutas Z, Martinsson P, et al. (2005) On the compression of low rank matrices. *SIAM Journal on Scientific Computing* 26(4): 1389–1404.

Dheeru D and Karra Taniskidou E (2017) UCI machine learning repository. Available at: http://archive.ics.uci.edu/ml (accessed 2 December 2014).

Feng Y, Xiao J and Gu M (2019) Flip-flop spectrum-revealing QR factorization and its applications to singular value decomposition. *Electronic Transactions on Numerical Analysis* 51: 469–494.

Foster LV (1997) The growth factor and efficiency of Gaussian elimination with rook pivoting. *Journal of Computational and Applied Mathematics* 86(1): 177–194.

Grasedyck L and Hackbusch W (2003) Construction and Arithmetics of H-matrices. *Computing* 70(4): 295–334.

Grasedyck L and Hackbusch W (2005) Hybrid cross approximation of integral operators. *Numerische Mathematik* 101(2): 221–249.

Gu M and Eisenstat S (1996) Efficient algorithms for computing a strong rank-revealing QR factorization. *SIAM Journal on Scientific Computing* 17(4): 848–869.

Hackbusch W, Grasedyck L and Börm S (2002) An introduction to hierarchical matrices. *Mathematica Bohemica* 127(2): 229–241.

Heldring A, Ubeda E and Rius JM (2014) On the convergence of the ACA algorithm for radiation and scattering problems. *IEEE Transactions on Antennas and Propagation* 62(7): 3806–3809.

Heldring A, Ubeda E and Rius JM (2015) Stochastic estimation of the Frobenius norm in the ACA convergence criterion. *IEEE Transactions on Antennas and Propagation* 63(3): 1155–1158.

Liberty E, Woolfe F, Martinsson PG, et al. (2007) Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences* 104(51): 20167–20172.

Mahoney MW and Drineas P (2009) CUR matrix decompositions for improved data analysis. *Proceedings of the National Academy of Sciences* 106(3): 697–702.

Martinsson PG, Quintana-Ortí and Heavner N (2019) randUTV: a blocked randomized algorithm for computing a rank-revealing UTV factorization. *ACM Transactions on Mathematical Software* 45(1): 1–26.

Musco C and Musco C (2017) Recursive sampling for the Nyström method. In: *Advances in Neural Information*

Processing Systems 30. New York: Curran Associates, Inc., pp. 3833–3845.

Neal L and Poole G (1992) A geometric analysis of Gaussian elimination. II. *Linear Algebra and its Applications* 173: 239–264.

Poole G and Neal L (1991) A geometric analysis of Gaussian elimination. I. *Linear Algebra and its Applications* 149: 249–272.

Poole G and Neal L (1992) Gaussian elimination: When is scaling beneficial? *Linear Algebra and its Applications* 162–164: 309–324.

Rebrova E, Chavez G, Liu Y, et al. (2018) A study of clustering techniques and hierarchical matrix formats for kernel ridge regression. *2018 IEEE international parallel and distributed processing symposium workshops (IPDPSW)*, Vancouver, Canada, 21–25 May 2018, pp. 883–892.

Voronin S and Martinsson PG (2017) Efficient algorithms for CUR and interpolative matrix decompositions. *Advances in Computational Mathematics* 43(3): 495–516.

Wang R, Li Y and Darve E (2018) On the numerical rank of radial basis function kernels in high dimensions. *SIAM Journal on Matrix Analysis and Applications* 39(4): 1810–1835.

Xiao J, Gu M and Langou J (2017) Fast parallel randomized QR with column pivoting algorithms for reliable low-rank matrix approximations. In: *2017 IEEE 24th international conference on high performance computing (HiPC)*, pp. 233–242. DOI: 10.1109/HiPC.2017.00035.

Zhao K, Vouvakis MN and Lee JF (2005) The adaptive cross approximation algorithm for accelerated method of moments computations of EMC problems. *IEEE Transactions on Electromagnetic Compatibility* 47(4): 763–773.

Zhou H, Zhu G and Kong W, et al. (2017) An upgraded ACA algorithm in complex field and its statistical analysis. *IEEE Transactions on Antennas and Propagation* 65(5): 2734–2739.

Author biographies

Yang Liu is a research scientist in the Scalable Solvers Group of the Computational Research Division at Lawrence Berkeley National Laboratory, in Berkeley, California. Dr Liu received the PhD degree in electrical engineering from the University of Michigan in 2015. From 2015 to 2017, he worked as a postdoctoral fellow at the Radiation Laboratory, University of Michigan. From 2017 to 2019, he worked as a postdoctoral fellow at Lawrence Berkeley National Laboratory, in Berkeley, California. His main research interest is in computational electromagnetics (including fast time-domain integral equation solvers, fast direct integral and differential equation solvers, and multi-physics modeling), numerical linear and multilinear algebras (including sparse solvers, randomized low-rank, butterfly and tensor algebras), and high-performance scientific computing. Dr Liu authored and co-authored the Sergei A. Schelkunoff Transactions Prize Paper, APS 2018, second place student paper, ACES 2012, and the first place student paper, FEM 2014.

Wissam Sid-Lakhdar is a postdoctoral researcher in the Scalable Solvers Group of the Computational Research Division at Lawrence Berkeley National Laboratory (LBL). He is currently working on the development of autotuning algorithms and software supported by the Exascale Computing Project (ECP). Before joining LBL, he was a postdoctoral fellow in the PARASOL laboratory in the Computer Science and Engineering Department of Texas A&M University. He was then working on autotuning batched QR factorization kernels on GPUs. He obtained his PhD at Ecole Normale Superieur of Lyon, where his work targeted the scalability of sparse linear algebra methods on heterogeneous architectures.

Elizaveta Rebrova is an Assistant Adjunct Professor in the Department of Mathematics of the University of California in Los Angeles. In Summers 2017 and 2018, she worked on the interplay between machine learning and numerical linear algebra in the Scalable Solvers Group of the Computational Research Division at Lawrence Berkeley National Laboratory. She received the PhD degree in mathematics from the University of Michigan in 2018. Her main research interests are high-dimensional probability and random matrix theory, and their applications to high-dimensional data science and linear algebra.

Pieter Ghysels is a research scientist in the Scalable Solvers Group of the Computational Research Division at Lawrence Berkeley National Laboratory, in Berkeley, California. His main interests are in High Performance Computing (HPC) and linear algebra. Pieter has expertise in both iterative methods and direct methods for the solution of systems of linear equations. He is the main developer of the STRUMPACK software library which offers a direct solver and preconditioners for large sparse linear systems as well as memory efficient representations of structured dense matrices. Pieter Ghysels received an engineering degree (in 2006) and completed a PhD in engineering Sciences, both at the (Flemish) Catholic University in Leuven, Belgium. From 2010–2013, Pieter worked at the Universiteit Antwerpen (University of Antwerp, Belgium) and at the Intel Exascience Lab Flanders.

Xiaoye Sherry Li is a Senior Scientist in the Computational Research Division, Lawrence Berkeley National Laboratory. She has worked on diverse problems in high performance scientific computations, including parallel computing, sparse matrix computations, high precision arithmetic, and combinatorial scientific computing. She has (co)authored over 110 publications, and contributed to several book chapters. She is the lead developer of SuperLU, a widely used sparse direct solver, and has contributed to the development of several other mathematical libraries, including ARPREC, LAPACK, PDSLin, STRUMPACK, and XBLAS. She has collaborated with many domain scientists to deploy the advanced mathematical software in their application codes, including those from accelerator engineering, chemical science, earth science, plasma fusion energy science, and materials science. She earned PhD in Computer Science from UC Berkeley in 1996. She has served on the editorial boards of the *SIAM Journal of Scientific Computing* and *ACM Transactions on Mathematical Software*, as well as many program committees of the scientific conferences. She is a SIAM Fellow and an ACM Senior Member.