Lawrence Berkeley National Laboratory
Recent Work

Title
A Parallel Hierarchical Blocked Adaptive Cross Approximation Algorithm

Permalink
https://escholarship.org/uc/item/2df365sj

Authors
Liu, Yang
Sid-Lakhdar, Wissam
Rebrova, Elizaveta
et al.

Publication Date
2019-09-17

Peer reviewed
ARTICLE TYPE

A Parallel Hierarchical Blocked Adaptive Cross Approximation Algorithm

Yang Liu*1 | Wissam Sid-Lakhdar1 | Elizaveta Rebrova2 | Pieter Ghysels1 | Xiaoye Sherry Li1

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

Correspondence
*Yang Liu, Email: liuyangzhuan@lbl.gov

Summary

This paper presents a low-rank decomposition algorithm assuming any matrix element can be computed in $O(1)$ time. The proposed algorithm first computes rank-revealing decompositions of sub-matrices 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 full decompositions such as rank-revealing QR decompositions. Numerical results demonstrate the efficiency, accuracy and parallel efficiency of the proposed algorithm.

KEYWORDS:
Adaptive cross approximation, singular value decomposition, rank-revealing decomposition, parallelization, multi-level 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, and analyzing numerical algorithms, etc, 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. This complexity can be reduced to $O(n^2\log r + nr^2)$ by structured random matrix projection-based algorithms. 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) algorithms can achieve $O(nr^2)$ complexity. However, the robustness of these algorithms relies heavily on the 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) 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 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). 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 constructs a low-rank factorization with optimal complexity upon accessing one row and column per iteration within typically $O(r)$ iterations. Despite its favorable computational complexity, it is well-known that the partially-pivoted ACA algorithms suffer from deteriorated convergence and/or premature termination for non-smooth, sparse and/or coherent matrices\textsuperscript{14}. Hybrid methods or improved convergence criteria (e.g., hybrid ACA-CUR, averaging, statistical norm estimation) have been proposed to partially alleviate the problem\textsuperscript{15,16}. 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 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 with 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. 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 $n_g$ submatrices compressed via BACA, followed by a hierarchical merge algorithm leveraging low-rank arithmetics\textsuperscript{17,18}. The overall cost of this H-BACA algorithm is at most $O(\sqrt{n_g}n_r^2)$ assuming the block size in BACA less than the rank and the resulting decomposition can be treated as a truncated SVD. In other words, the proposed H-BACA algorithm is a general numerical linear algebra tool as an alternative to ACA, SVD, QR, etc. In addition, the overall algorithm can be parallelized using the distributed-memory linear algebra packages such as ScaLAPACK\textsuperscript{20}. 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\textsuperscript{20}.

2  | NOTATION

Throughout this paper, 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 $I, J$ 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 $A^T$, $A^{-1}$, $A^{+}$. $\|A\|_F$ and $\|u\|_2$ denote Frobenius norm and 2-norm. Note that $u$ refers to a $n \times 1$ column vector. Row-wise and column-wise concatenations of $A, B$ are $[A; B]$ and $[A, B]$. Element-wise multiplication of $A$ and $B$ is $A \times B$. 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 $[U, \Sigma, V, r] = \text{SVD}(A, \epsilon)$ 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 $\epsilon$-rank defined by $r = \min \{k \in \mathbb{N} : \Sigma_{k+1,k+1} < \epsilon \Sigma_{1,1}\}$. We denote QRCP as $[Q, T, J] = \text{QR}(A, \epsilon)$ or $[Q, T, J] = \text{QR}(A, \epsilon)$ with $Q \in \mathbb{R}^{m \times n}$ column orthogonal, $T \in \mathbb{R}^{n \times r}$ upper triangular, $J$ being column pivots, and $\epsilon$ and $r$ being the prescribed accuracy and rank, respectively. QR without column-pivoting is simply written as $[Q, T] = \text{QR}(A)$. Cholesky decomposition without pivoting is written as $T = \text{Chol}(A)$ 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\textsuperscript{8}. Consider a matrix $A \in \mathbb{R}^{m \times n}$ of $\epsilon$-rank $r$, the ACA algorithm approximates $A$ by a sequence of rank-1 outer-products as

$$A \approx UV = \sum_{k=1}^{r} u_k v_k^T$$  \hspace{1cm} (1)

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 $E_{k-1} = A - \sum_{i=1}^{k-1} u_i v_i^T$ corresponding to the largest element in magnitude denoted by $E_{k-1}(i_k, j_k)$.
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, the pivot pair \((i_k, j_k)\) at iteration \( k \) is selected (via line 4 and 7) as

\[
j_k = \arg \max_j |E_{k-2}(i_{k-1}, :)|, j \neq j_1, ..., j_{k-1}, k > 1
\]

\[
i_k = \arg \max_i |E_{k-1}(:)i|, i \neq i_1, ..., i_{k-1}
\]

and \( j_1 \) is a random initial column index. The iteration is terminated when \( \nu < \epsilon \mu \) with

\[
\nu = \|u_k v_k^T\|_F, \quad \mu = \|UV\|_F \approx \|A\|_F
\]

and \( \epsilon \) is the prescribed tolerance. Note that each iteration requires only \( O(nr_k) \) flop operations with \( r_k \) denoting the current iteration number (and 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 in the numerical solution of PDEs, signal processing and data science, ACA oftentimes either exhibits premature termination or requires \( O(n) \) iterations. Remedies such as averaged stopping criteria,\(^{21}\) stochastic error estimation,\(^{12}\) \( ACA^+\),\(^ {16}\) and hybrid ACA\(^ {16}\) have been developed but they do not generalize to a broad range of applications.

### 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 \( U = 0, V = 0, \mu = 0, j_1 \) is a random column index;
2 for \( k = 1 \) to \( \min\{m, n\} \) do
3 \( \quad u_k = A(:, j_k) - UV(:, j_k) \);
4 \( \quad i_k = \arg \max_i |u_k(i)|, i \neq i_1, ..., i_{k-1} \);
5 \( \quad u_k \leftarrow u_k / u_k(i_k) \);
6 \( \quad v_k^T = A(i_k, :) - U(i_k, :)V \);
7 \( \quad j_{k+1} = \arg \max_j |v_k(j)|, j \neq j_1, ..., j_k \);
8 \( \quad \nu^2 = \|u_k\|_2^2 \|v_k\|_2^2 \);
9 \( \quad \mu^2 \leftarrow \mu^2 + \nu^2 + 2 \sum_{j=1}^{k} V(j, :)v_k u_k^T U(:, j) \);
10 \( \quad U \leftarrow [U, u_k], V \leftarrow [V, v_k^T] \), \( r_k = r_{k-1} + 1 \);
11 Terminate if \( \nu < \epsilon \mu \).
```

### 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 hence higher flop performance can be achieved.

Specifically, the proposed BACA algorithm factorizes \( A \) as

\[
A \approx UV = \sum_{k=1}^{r_d} U_k V_k
\]

where \( U_k \in \mathbb{R}^{m \times d_k} \) and \( V_k \in \mathbb{R}^{d_k \times n} \) with block size \( d \), \( d_k \approx d \) and the number of blocks \( r_d \approx r/d \). Instead of selecting row/column pivots via line 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.
Algorithm 2: Blocked adaptive cross approximation algorithm (BACA)

\textbf{input}: Matrix } A \in \mathbb{R}^{m \times n}, \text{ block size } d, \text{ relative tolerance } \epsilon \\
\textbf{output}: Low-rank approximation of } A \approx UV \text{ with rank } r \\
1 \quad U = 0, \quad V = 0, \quad r = 0, \quad \mu = 0, \quad J_1 \text{ is a random index set of cardinality } d; \\
2 \quad \textbf{for } k = 1 \textbf{ to } \min\{m, n\}/d \textbf{ do} \\
3 \quad \quad C_k = A(:, J_k) - UV(:, J_k); \\
4 \quad \quad \hat{C}_k(i, :) = C_k(i, :) \text{ for } i \notin \bigcup_{s=1}^{k-1} I_s \text{ and zero elsewhere;} \\
5 \quad \quad [Q_k^c, T_k^c, I_k] = \text{QR}(\hat{C}_k, d); \\
6 \quad \quad R_k = A(I_k, :) - U(I_k, :)V; \\
7 \quad \quad R_k(:, j) = R_k(:, j) \text{ for } j \notin \bigcup_{s=1}^{k-1} J_s \text{ and zero elsewhere;} \\
8 \quad \quad [Q'_{k+1}, T'_{k+1}, J_{k+1}] = \text{QR}(R_k, d); \\
9 \quad \quad W_k = A(I_k, J_k) - U(I_k, :)V(:, J_k); \\
10 \quad \text{[U}_k, V_k, d_k, J] = \text{LRID}(C_k, W_k, R_k); \\
11 \quad \quad I_k \leftarrow I_k([1, d_k]), J_k \leftarrow J_k(J); \\
12 \quad \quad r_k = r_{k-1} + d_k; \\
13 \quad \quad \nu = \text{LRnorm}(U_k, V_k); \\
14 \quad \quad \mu \leftarrow \text{LRnormUp}(U, V, \mu, U_k, V_k, \nu); \\
15 \quad \quad U \leftarrow [U, U_k], V \leftarrow [V; V_k]; \\
16 \quad \quad \text{Terminate if } \nu < \epsilon \mu. \\
17 \quad \textbf{Function LRID}(C, W, R, \epsilon) \\
\text{\textbf{input}: } C = A(:, J), R = A(I, :) \text{ with } I, J \text{ of same cardinality} \\
\text{\textbf{output}: } A \approx UV \text{ with } U \in \mathbb{R}^{m \times r}, V \in \mathbb{R}^{r \times n} \\
18 \quad \quad [Q, T, \tilde{J}, r] = \text{QR}(W, \epsilon); \\
19 \quad \quad U = C(:, \tilde{J}); \\
20 \quad \quad V = T^{-1}Q'R; \\
21 \quad \quad \textbf{return } U, V, r, \tilde{J}. \\
22 \quad \textbf{Function LRnorm}(U, V) \\
\text{\textbf{input}: } A = UV \\
\text{\textbf{output}: } \|A\|_F \\
23 \quad \quad T_1 = \text{Cho1}(U^T U); \\
24 \quad \quad T_2 = \text{Cho1}(V V^T); \\
25 \quad \quad \textbf{return } \|T_1 T_2^T\|_F; \\
26 \quad \textbf{Function LRnormUp}(U, V, \nu, \tilde{U}, \tilde{V}, \tilde{\nu}) \\
\text{\textbf{input}: } U \in \mathbb{R}^{m \times \tilde{r}}, V \in \mathbb{R}^{r \times \tilde{r}}, \tilde{U} \in \mathbb{R}^{m \times \tilde{r}}, \tilde{V} \in \mathbb{R}^{r \times \tilde{r}}, \nu = \|UV\|_F, \tilde{\nu} = \|\tilde{U}\tilde{V}\|_F \\
\text{\textbf{output}: } \|[U, \tilde{U}][V; \tilde{V}]\|_F \\
27 \quad \quad s = \tilde{v}^2 + \tilde{v}^2 + 2 \sum_{i=1}^{\tilde{r}} \sum_{j=1}^{\tilde{r}} \tilde{V}(i, j) \text{ with } \tilde{V} = (V \tilde{V})\sigma(U^T \tilde{U}); \\
28 \quad \quad \textbf{return } \sqrt{s} \\

- Find block row } I_k \text{ and block column } J_{k+1} \text{ by QRCP. Starting with a random column index set } J_1, \text{ the block row } I_k \text{ and} \\
\text{the next iteration’s block column } J_{k+1} \text{ are selected by (line 5 and 8)} \\
\begin{align*} 
[Q_k^c, T_k^c, I_k^c] &= \text{QR}(E_{k-1}^c(I_k^c, J_k), d), \quad I_k = [1, m] \setminus \bigcup_{s=1}^{k-1} I_s \\
I_k &= I_k^c(I_k^c) \\
\end{align*} \\
\begin{align*} 
[Q'_{k+1}, T'_{k+1}, J_{k+1}'] &= \text{QR}(E_{k-1}^c(I_k, J_{k+1}'), d), \quad J_{k+1} = [1, n] \setminus \bigcup_{s=1}^{k} J_s \\
J_{k+1} &= J_{k+1}'(J_{k+1}') \\
\end{align*}
Here the algorithm first selects $d$ skeleton rows from the submatrix $E_{k-1}(J_k, I_k^c)$ (i.e., $d$ columns from its transpose) and then selects $d$ skeleton columns from the submatrix $E_{k-1}(I_k, J_k^c)$ by leveraging QRCP as it provides a natural way of sorting columns by their norms during the factorization. Note that $I_k \subseteq I_k^c$ and $J_k \subseteq J_k^c$ are enforced to avoid selecting previously selected rows and columns. See Fig. [4][a] for an illustration of the procedure. $I_k$ and $J_k$ are selected by QRCP on the column 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 Fig. [4][a] consist of contiguous indices.

- Form the factors of the low-rank product $U_{k}V_{k}$. Let $C_k = E_{k-1}(\cdot, J_k)$, $R_k = E_{k-1}(I_k, \cdot)$ and $W_k = E_{k-1}(I_k, J_k)$, $E_{k-1}$ can be approximated by an ID-type decomposition $E_{k-1} \approx C_k W_k^{-1} R_k = U_{k}V_{k}^{-1}$ by (8) and (9). Note that the pseudo inverse is computed via rank-revealing QR (also see the LRID at line[10]). The rank-revealing algorithm is needed as the $d \times d$ block $W_k$ can be further compressed with rank $d_k$. Consequently, the effective rank increase is $d_k \leq d$ and the pivot pair $(I_k, J_k)$ is updated in (10) by the column pivots $\tilde{J}$ of QRCP in (8).

\[
\begin{align*}
\{Q, T, \tilde{J}, d_k\} &= \text{QR}(W_k, \epsilon) \\
U_k &= C_k(\cdot, \tilde{J}), V_k = T^{-1}Q'R_k \\
I_k &\leftarrow I_k([1, d_k]), J_k \leftarrow J_k(\tilde{J})
\end{align*}
\]

- Compute $\nu = \|U_kV_k\|_F$ 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 [3][a]) using

\[
\begin{align*}
T_{U_k} &= \text{Chol}(U_k'U_k), T_{V_k} = \text{Chol}(V_k'V_k) \\
\nu &= \|T_{U_k}T_{V_k}'\|_F
\end{align*}
\]

Once $\nu$ is computed, the norm of $UV$ can be updated efficiently in $O(nr_kd_k)$ operations (line [4][a]) as

\[
\mu^2 \leftarrow \mu^2 + \nu^2 + 2 \sum_{i=1}^{r_k} \sum_{j=1}^{d_k} \tilde{V}(i, j) \text{ with } \tilde{V} = (VV_k') o(UU_k)
\]

where $r_k$ represents the column dimension of $U$ at iteration $k$. Note that the matrix multiplications in (11) and (13) involving $V_k$ and $V$ (and similarly for those involving $U_k$ and $U$) can be performed as $[V, V_k]V_k'$ to further improve the computational efficiency. Then the algorithm updates $U, V$ as $[U, U_k], [V, V_k]$ and tests the stopping criteria $\nu < \epsilon \mu$.

We would like to highlight the difference between the proposed BACA algorithm and the 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[20] 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 suffers from poor flop performance. Second, BACA also has important connections to the hybrid ACA algorithm[26]. 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 interpolation algorithms (LRID at line [10]) in each iteration and uses QRCP to select skeleton rows and columns. 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 tradeoff 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 $O(nr_kd)$ operations. For example, BACA (Algorithm 2) precisely reduces to ACA (Algorithm 1) when $d = 1$. In what follows we refer to the ACA algorithm as BACA when $d = 1$. 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 [10] remains the only nontrivial operation). In this case the algorithm requires $O(nr^2)$ 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, a SVD re-compression of $U$ and $V$ may be needed via first computing a QR of $U$ and $V$ as $[Q_U, T_U] = \text{QR}(U), [Q_V, T_V] = \text{QR}(V)$,
Hierarchical Low-Rank Merge

For simplicity, it is assumed that dividing each index set into \( k \)-level binary trees for index vectors \( A \) and \( B \) respectively. The leaf and root levels are denoted \( 0 \) and \( L \) respectively.

The proposed BACA algorithm can be further enhanced with a hierarchical low-rank merge algorithm leveraging low-rank -level binary trees for index vectors \( A \) and \( B \) respectively. The results can be viewed as a truncated SVD of \( A \) and we assume this is the output of the BACA algorithm in the rest of this paper.

**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 \( I_i \) and \( J_i \) for nodes \( \tau \) and \( \nu \) at each level, \( L = \log_2 n_b \), 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.         \([U_{\tau \nu}, \Sigma_{\tau \nu}, V_{\tau \nu}, r_{\tau \nu}] = \text{BACA}(A_{\tau \nu}, d, \epsilon)\);
      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.           \( \tilde{U}_{\tau \nu} = [U_{\tau_{i1}}, \Sigma_{\tau_{i1}}, U_{\tau_{i2}}, \Sigma_{\tau_{i2}}]; \)
            10.          \( \tilde{V}_{\tau \nu} = \text{diag}(V_{\nu_{i1}}, V_{\nu_{i2}}); \)
            11.          \([U_{\tau \nu}, \Sigma_{\tau \nu}, V_{\tau \nu}, r_{\tau \nu}] \leftarrow \text{SVD}(\tilde{U}_{\tau \nu}, \epsilon)\);
            12.          \( \tilde{V}_{\nu} = \text{diag}(V_{\nu_{i1}}, V_{\nu_{i2}}); \)
            13.          \( \tilde{V}_{\tau \nu} = [\Sigma_{\tau_{i1}}V_{\nu_{i1}}, \Sigma_{\tau_{i2}}V_{\nu_{i2}}]; \)
            14.          \([U_{\tau \nu}, \Sigma_{\tau \nu}, V_{\tau \nu}, r_{\tau \nu}] \leftarrow \text{SVD}(\tilde{V}_{\tau \nu}, \epsilon)\);
            15.          \( U_{\tau \nu} \leftarrow \tilde{U}_{\nu \nu}, V_{\tau \nu} \)
```

3.3 Hierarchical Low-Rank Merge

The proposed BACA algorithm can be further enhanced with a hierarchical low-rank merge algorithm leveraging low-rank arithmetics to achieve improved robustness and parallelism. 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 \( I_i \) and \( J_i \) for nodes \( \tau \) and \( \nu \) at each level, upon recursively dividing each index set into \( I_{\tau j} \) of approximately equal sizes, \( i = 1, 2, j = 1, 2 \). Here, \( \tau_i \) and \( \nu_j \) are children of \( \tau \) and \( \nu \), 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 \( \tau, \nu \) as \( A_{\tau \nu} = A(I_{\tau}, J_{\nu}) \) and their truncated SVD as \([U_{\tau \nu}, \Sigma_{\tau \nu}, V_{\tau \nu}, r_{\tau \nu}] = \text{SVD}(A_{\tau \nu}, \epsilon)\). Here \( r_{\tau \nu} \) is the \( \epsilon \)-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 $A_{r_k}$ 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 arithmetics. 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. Fig. 15 illustrates one re-compression operation for transforming SVDs of $A_{r_{k+1}}$, $i = 1, 2$, $j = 1, 2$ into that of $A_{r_k}$. The operation first column-wise compresses SVDs of $A_{r_{k+1}}$, $i = 1, 2$, $j = 1, 2$ at step $l = \frac{1}{2}$ and then row-wise compresses the results, i.e., SVDs of $A_{r_{k+1}}$, $i = 1, 2$ at step $l$, $l = 1, \ldots, L$. Specifically, the column-wise compression step is composed of one concatenation operation in (14) and one compression operation in (15):

$$\bar{U}_{r_{k+1}} = [U_{r_{k+1}} \Sigma_{r_{k+1}}, U_{r_{k+1}} \Sigma_{r_{k+1}}], \bar{V}_{r_{k+1}} = \text{diag}(V_{r_{k+1}}, V_{r_{k+1}})$$

(14)

$$[U_{r_{k}}, \Sigma_{r_{k}}, V_{r_{k}}, r_{k}] \leftarrow \text{SVD}(\bar{U}_{r_{k+1}}, \epsilon), V_{r_{k}} \leftarrow V_{r_{k+1}} \bar{V}_{r_{k+1}}$$

(15)

with $i = 1, 2$. Let $\bar{U}_{r_{k+1}}$, $\bar{V}_{r_{k+1}}$, and $U_{r_{k}}, \Sigma_{r_{k}}, V_{r_{k}}$ denote the submatrix before and after the SVD truncation, respectively. Similarly, the row-wise compression step can be performed via column-wise merge of $A_{r_{k+1}}$, $i = 1, 2$. Let $r_l$ represent the maximum rank $r_{k+1}$ among all blocks at steps $l = 0, 1, \ldots, L$. Note the algorithm returns a 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 Fig. 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 column-wise pair of blocks; similarly at level $l = 1, 2$, the algorithm merges every row-wise 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 $m = n = 2^l$ and $p = 2^l$. The proposed parallel implementation first creates two $\lceil \log \sqrt{p} \rceil$-level binary trees with $p$ denoting the total number of MPI processes. One process performs BACA compression of one or two leaf-submatrix 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 ScALAPACK with process grids that aggregate those in corresponding submatrices. Consider the example in Fig. 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 ScALAPACK grids of $2 \times 1, 2 \times 2$, and $4 \times 2$, respectively. For illustration purpose, we select the ScALAPACK block size in Fig. 2 as $n_0 \times n_0$ where $n_0$ is the dimension of the finest-level submatrices in the hierarchical merge algorithm and $n = \sqrt{n_0 n_0}$. 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 $m = n = 2^l$ and $p = 2^l$ is not needed in practice.

4 1 COST ANALYSIS

In this section, the computational costs of the proposed BACA and H-BACA algorithms are analyzed. First, the costs for BACA can be summarized as follows. Assuming BACA converges in $O(f/d)$ 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
Table 1: Flop counts for the leaf-level compression and hierarchical merge operation in Algorithm 3 for two classes of low-rank matrices with different block sizes $d$

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>ACA/ACA$^+$</th>
<th>Hyrbird-ACA</th>
<th>BACA</th>
<th>H-BACA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pivot count per iteration</td>
<td>1</td>
<td>1</td>
<td>$d$</td>
<td>$n_0d$</td>
</tr>
<tr>
<td>Cost (constant rank)</td>
<td>$O(nr^2)$</td>
<td>$O(nr^2)$</td>
<td>$O(nr^2)$</td>
<td>$O(nr^2 \sqrt{n_b})$</td>
</tr>
<tr>
<td>Cost (increasing rank)</td>
<td>$O(nr^2)$</td>
<td>$O(nr^2)$</td>
<td>$O(nr^2)$</td>
<td>$O(nr^2)$</td>
</tr>
<tr>
<td>Pre-selection of submatrices</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 2: Comparisons between proposed BACA, H-BACA algorithms with existing ACA algorithms. Note that the algorithms show increasing robustness from left to right.

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

$$c_{BACA} = O(\frac{r}{d}) \sum_{k=1}^{O(\log d)} (nd^2 + nr_kd + d_kd^2) \leq O(nd^2 + rd^2 + nrdO(\frac{r}{d}))$$

(16)

When $d < r$ (e.g., $d = O(1)$), it can be validated that $c_{BACA} = O(nr^2)$; when $d \gg r$ (e.g., $d = O(n)$), it follows that $c_{BACA} = O(n^2r)$ assuming that the $nd^2$ term due to QRCP vanishes as the pivot selection can be bypassed.

Next, the computational costs of the H-BACA algorithm are analyzed. The costs are analyzed for two cases of distributions of the maximum ranks $r_l$ at each level, i.e., $r_l = r$ (ranks stay constant during the merge) and $r_l \approx 2^l r/\sqrt{n_b} = 2^{l-1} r$ (rank increases by a factor of 2 per level), $l = 0, 1, ..., L$. The constant-rank case is often valid for matrices with their numerical rank 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 rank is a constant proportion of 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). From the above-described analysis of BACA, the computational costs for the leaf-level compression $c_b = c_{BACA}n_b$ are:

$$c_b = \frac{n}{\sqrt{n_b}} n_b^2 r_0^2, \text{ if } d \leq r_0$$

(17)

$$c_b = \left(\frac{n}{\sqrt{n_b}}\right)^2 r_0 n_b, \text{ if } d = O(n/\sqrt{n_b})$$

(18)

which represent the complexity with ACA and QRCP when $d = 1$ and $d = n/\sqrt{n_b}$, respectively.

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

$$c_m = \sum_{l=1}^{L} 4^{L-l} n_l r_l^2$$

(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 2. Note that the costs of the BACA algorithm can also be extracted from Table 2 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 an $1/\sqrt{n_b}$ reduction factor for the increasing rank case and $\sqrt{n_b}$ overhead for the constant rank case.

For completeness, the comparison between the proposed BACA, H-BACA algorithms (assuming $d \leq r_b$) 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 $n_b d$ 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 $\sqrt{n_b}$.

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: $A_{i,j} = \exp(-\|x_i-x_j\|^2)$, $i, j = 1, \ldots, 2n$. Here $h$ is the Gaussian width, and $x_i, x_j \in \mathbb{R}^{8\times 1}$ 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, respectively. Note that the Gaussian kernel permits low-rank compression as shown in 22–25.

2. Helmholtz kernel: $A_{i,j} = H_0^{(2)}(k \|x_i-x_j\|)$. 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. Note that $A$ is a complex-valued matrix.

3. 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 dataset, and $h$ is a regularization parameter.

4. ToeplitzQchem kernel: $A_{i,j} = \frac{(-1)^{i+j}}{(i-j)!}$. 5. Product-of-random kernel: $A = U V^T$ with $U \in \mathbb{R}^{r\times r}$ and $V \in \mathbb{R}^{r\times r}$ being random matrices with i.i.d. entries. Throughout this section, we refer to ACA and QRCP as special cases of BACA when $d = 1$ and $d = n/\sqrt{n_b}$, respectively. In all examples except for the Product-of-random kernel, the algorithm is applied to the off-diagonal submatrix $A_{12} = A(1 : n, 1 + n : 2n)$ assuming rows/columns of $A$ have been properly permuted. Note that the permutation may yield a hierarchical matrix representation of $A$, but in this paper we only focus on 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 E5-2698 v3 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 multiple matrices: Gaussian-SUSY matrices with $n = 5000$, $h = 1.0, 0.2$, Polynomial matrices with $n = 10000$, $h = 0.2$, and Helmholtz matrices with $n = 20000$. The corresponding $\epsilon$-ranks are $r = 4683, 1723, 1293, 302$ for $\epsilon = 10^{-6}$. The convergence histories of BACA with $d = 1, 32, 64, 128, 256$ and $n$ are plotted in Fig. 3. The residual error for $d < n$ is defined as $\|U V_k\|_F / \|U V\|_F$ from (12). For $d = 1$, $32, 64, 128, 256$, the iteration number is multiplied with $d$ for $d < n$ to reflect the true convergence performance, as BACA picks $d$ column/rows per iteration. For $d = n$, the convergence history of QRCP in LRID is plotted. The residual error for QRCP is defined as $T(k, k)/T(1, 1)$ with $[Q, T, J] = Q\mathcal{R}(A, \epsilon)$.

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, the residual exhibits wild oscillations and even causes premature iteration termination for $h = 0.2$ (see Fig. 3b). The QRCP algorithm ($d = n$), in stark contrast, achieves the desired accuracy after approximately $r$ iterations (requiring $O(n^3)$ operations per iteration though). The proposed BACA algorithm ($d = 32, 64, 128, 256$) shows increasingly smooth residual histories as $d$ increases. For the Polynomial (Fig. 3c) and Helmholtz (Fig. 3d) matrices, BACA also shows better convergence behaviors compared to ACA with even small block sizes $d > 1$.

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 Polynomial matrix with $n = 10000$, $h = 0.2$ and one Helmholtz matrices with $n = 5000$. The relative Frobenious-norm error $\|A - U V\|_F / \|A\|_F$ is computed via changing number of leaf-level submatrices $n_b$ and block size $d$. When $h = 1.0$ for the Gaussian-SUSY matrix (Fig. 3b), the H-BACA algorithms achieve desired accuracies ($\epsilon = 10^{-2}, 10^{-6}, 10^{-10}$) using the baseline ACA ($d = 1$), BACA ($d = 32$), QRCP ($d = n/\sqrt{n_b}$) when $n_b = 1$ and the hierarchical
merge operation only causes slight error increases as $n_b$ increases. Similar results have been observed for the Polynomial (Fig. 4c) and Helmholtz (Fig. 4d) matrices. When $h = 0.2$ for the Gaussian-SUSY matrix (Fig. 4b), H-BACA with QRCP still attains the desired accuracy for all data points while H-BACA with ACA fails. In comparison, the H-BACA with $d = 32$ is slightly better than $d = 1$ when $n_b = 1$ but the accuracy improves as $n_b$ increases (see the curves marked with “+” in Fig. 4c).

5.3 | Efficiency

This subsection provides several examples to verify the complexity estimates in Table 1. H-BACA with leaf-level ACA ($d = 1$), BACA ($d = 8, 16, 32, 64, 128$), and QRCP is tested for the following matrices: one Helmholtz matrix with $n = 40000, \epsilon = 10^{-4}$, one Gaussian-SUSY matrix with $n = 50000, h = 1.0, \epsilon = 10^{-2}$, one Gaussian-MNIST matrix with $n = 5000, h = 3.0, \epsilon = 10^{-2}$, one Polynomial matrix with $n = 10000, h = 0.2, \epsilon = 10^{-4}$, one ToepplitzQchem matrix with $n = 100000, \epsilon = 10^{-4}$, and one Product-of-random matrix with $n = 2500, \epsilon = 10^{-4}$. The corresponding $c$-ranks are 292, 298, 137, 450, 9, and 1000, respectively. It can be validated that the hierarchical merge operation attains increasing ranks for the Helmholtz and Gaussian matrices, and relatively constant ranks for the Polynomial, ToepplitzQchem 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 Fig. 5. Note that the data points where the algorithm fails are shown as triangular markers without lines.

For the algorithms with QRCP, Table 1 suggests that the CPU time stays constant w.r.t. $n_b$ when the hierarchical merge operation attains constant ranks $r_j$, which is partially observed for ToepplitzQchem and Product-of-random matrices (see the blue curves in Fig. 5c, 5d). Also, the factor of $1/\sqrt{n_b}$ reduction in CPU time when $r_j$ increases is also observed for the Gaussian and Helmholtz matrices (see the blue curves in Fig. 5a, 5b). For the algorithms with ACA and BACA, Table I predicts increasing (with a factor of $\sqrt{n_b}$) and constant time when $r_j$ stays constant and increases, respectively. For the Gaussian and Helmholtz matrices, we observe non-increasing CPU time w.r.t. $n_b$ when $n_b$ is not too big. For the Gaussian-MNIST matrices, the CPU time is even reduced due to improved BLAS performance (see Fig. 5c when $n_b \leq 16$). For the ToepplitzQchem, Polynomial and Product-of-random matrices, increasing CPU time w.r.t. $n_b$ is observed as predicted by Table I. For most data points, the CPU time of the algorithms with ACA and BACA outperforms that with QRCP.
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 Fig. 5b and \(d = 1, 8\) curves in Fig. 5c. In contrast, H-BACA with larger \(d\) attains desired accuracies. Second, larger block size \(d\) results in reduced CPU time for the Polynomial matrices due to better BLAS performance (see Fig. 5d). However, larger block sizes cause increasing CPU time for the ToeplitzQchem matrices due to overestimation of the rank (when \(d > r = 9\)) for corresponding submatrices (see Fig. 5e). For the other matrices, no significant performance differences have been observed by changing block size \(d\).

5.4 Parallel performance

Finally, the parallel performance of the H-BACA algorithm is demonstrated via strong scaling studies with the Helmholtz and Product-of-random matrices. For the Helmholtz matrices, \(n = 160000\) and the wavenumbers are chosen such that the \(c\)-ranks with \(c = 10^{-4}\) are \(r = 30, 450\) and 890, respectively. H-BACA with \(d = 1\) is tested with process count \(p = 4, ..., 1024\). For the Product-of-random matrices, \(n = 10000\) and the inner dimension of the product is set to \(r = 2000\) and 100, respectively. H-BACA with \(d = 8\) is tested with process count \(p = 16, ..., 1024\). In both tests, the number of leaf-level subblocks is chosen as \(\sqrt{n_k} = \lceil \sqrt{p} \rceil\). The ScaLAPACK block size is set to \(64 \times 64\). For small ranks (\(r = 30\) for Helmholtz and \(r = 100\) for Product-of-random), poor parallel efficiency is observed due to partially utilized process grids at each re-compression step and the computational overhead of \(\sqrt{n_k}\); for larger ranks (\(r = 450, 890\) for Helmholtz and \(r = 800, 2000\) for Product-of-random), good parallel efficiencies are achieved (see Fig. 6). Not surprisingly, the parallel runtime is dominated by that of ScaLAPACK computation and possible redistributions between each re-compression step. Note that the leaf-level BACA compression is embarrassingly parallel for all test cases.
FIGURE 5 CPU time of H-BACA for (a) Helmholtz, (b) Gaussian-SUSY, (c) Gaussian-MNIST, (d) Polynomial, (e) ToeplitzQchem, and (f) Product-of-random matrices with varying $n_h$.

6 | CONCLUSION

This paper presents a fast and robust low-rank matrix decomposition algorithm given that any matrix entry can be evaluated in $O(1)$ time. The proposed algorithm performs blocked adaptive cross approximation (BACA) algorithms on submatrices followed by a hierarchical low-rank merge algorithm. The BACA algorithm significantly improves the robustness of the baseline ACA algorithm and maintains low computational complexity. The H-BACA algorithm combines results of BACA into the desired low-rank decomposition to further increase robustness and parallelism. Analysis and numerical examples demonstrate favorable efficiency and accuracy of the proposed algorithm for broad ranges of matrices.

ACKNOWLEDGMENT

This research was supported in part by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration, and in part by the U.S. Department of
FIGURE 6 CPU time of H-BACA for (a) Helmholtz with $d = 1$ and (b) Product-of-random matrices with $d = 8$ with varying processor counts.

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.

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

References


