Parallel Hierarchical Clustering using Rank-Two Nonnegative Matrix Factorization

Lawton Manning and Grey Ballard
Wake Forest University
Winston-Salem, NC, USA
{mannlg15,ballard}@wfu.edu

Ramakrishnan Kannan
Oak Ridge National Laboratory
Oak Ridge, TN, USA
kannanr@ornl.gov

Haesun Park
Georgia Institute of Technology
Atlanta, GA, USA
hpark@cc.gatech.edu

***Abstract*—Nonnegative Matrix Factorization (NMF) is an effective tool for clustering nonnegative data, either for computing a flat partitioning of a dataset or for determining a hierarchy of similarity. In this paper, we propose a parallel algorithm for hierarchical clustering that uses a divide-and-conquer approach based on rank-two NMF to split a data set into two cohesive parts. Not only does this approach uncover more structure in the data than a flat NMF clustering, but also rank-two NMF can be computed more quickly than for general ranks, providing comparable overall time to solution. Our data distribution and parallelization strategies are designed to maintain computational load balance throughout the data-dependent hierarchy of computation while limiting interprocess communication, allowing the algorithm to scale to large dense and sparse data sets. We demonstrate the scalability of our parallel algorithm in terms of data size (up to 800 GB) and number of processors (up to 80 nodes of the Summit supercomputer), applying the hierarchical clustering approach to hyperspectral imaging and image classification data. Our algorithm for Rank-2 NMF scales perfectly on up to 1000s of cores and the entire hierarchical clustering method achieves 5.9x speedup scaling from 10 to 80 nodes on the 800 GB dataset.**

***Index Terms*—low-rank approximation, distributed-memory parallel algorithms, scalable clustering**

I. Introduction

Nonnegative Matrix Factorization (NMF) has been demonstrated to be an effective tool for unsupervised learning problems including clustering [4], [18], [21]. An NMF consists of two tall-and-skinny non-negative matrices whose product approximates a nonnegative data matrix. That is, given an $m \times n$ data matrix $\mathbf{A}$, we seek nonnegative matrices $\mathbf{W}$ and $\mathbf{H}$ that each have k columns so that $\mathbf{A} \approx \mathbf{W}\mathbf{H}^\mathsf{T}$. Each pair of corresponding columns of $\mathbf{W}$ and $\mathbf{H}$ form a latent component of the NMF. If the rows of $\mathbf{A}$ correspond to features and the columns to samples, the ith row of the $\mathbf{H}$ matrix represents the loading of sample i onto each latent component and provides a soft clustering. Because the $\mathbf{W}$ factor is also nonnegative, each column can typically be interpreted as a latent feature vector for each cluster.

Hierarchical clustering is the process of recursively paritioning a group of samples. While standard NMF is interpreted as a flat clustering, it can also be extended for hierarchical clustering. Kuang and Park [15] propose a method that uses rank-2 NMF to recursively bipartition the samples. The method determines a binary tree such that all leaves contain unique samples and the structure of the tree determines hierarchical clusters.A single $\mathbf{W}$ vector for each node can also be used for cluster interpretation. We discuss the hierarchical method in more detail in Section II and § III-A.

We illustrate the output of the hierarchical clustering method with an example data set and output tree. Following Gillis et al. [9], we apply the method to a hyperspectral imaging (HSI) data set of the Washington, D.C national mall, which has pixel dimensions 1280×307 and 191 spectral bands. Figure 1 visualizes the output tree with 6 leaves along with their hierarchical relationships. The root node, labeled 0, is a flattening of the HSI data to a 2D grayscale image. Each other node is represented by an overlay of the member pixels of the clusters (in blue) on the original grayscale image. The first bipartitioning separates vegetation (cluster 1) from non-vegetation (cluster 2), the bipartitioning of cluster 1 separates grass (cluster 3) from trees (cluster 4), the bipartitioning of cluster 2 separates buildings (cluster 5) from sidewalks/water (cluster 6), and so on. If the algorithm continues, it chooses to split the leaf node that provides the greatest benefit to the overall tree, which can be quantified as a node's "score" in various ways.

While the hierarchical clustering method offers advantages in terms of interpretation as well as execution time compared to flat NMF, implementations of the algorithm are limited to single workstations and the dataset must fit in the available memory. Currently available implementations can utilize multiple cores via MATLAB [15] or explicit shared-memory parallelization in the SmallK library [5].

The goal of this work is to use distributed-memory parallelism to scale the algorithm to large datasets that require the memory of multiple compute nodes and to high processor counts. While flat NMF algorithms have been scaled to HPC platforms [2], [8], [12], [17], our implementation is the first to our knowledge to scale a hierarchical NMF method to 1000s of cores. As discussed in detail in § III-B, we choose to parallelize the computations associated with each node in the tree, which involve a Rank-2 NMF and the computation of the node's score. We choose a data matrix distribution across processors that avoids any redistribution of the input matrix regardless of the data-dependent structure of the tree's splitting decisions so that the communication required involves only the small factor matrices. Analysis of the algorithm shows the dependence of execution time on computation and communication costs as well as on k, the number of clusters computed. In particular, we confirm that many of the dominant costs are logarithmic in k, which is favorable to the linear or

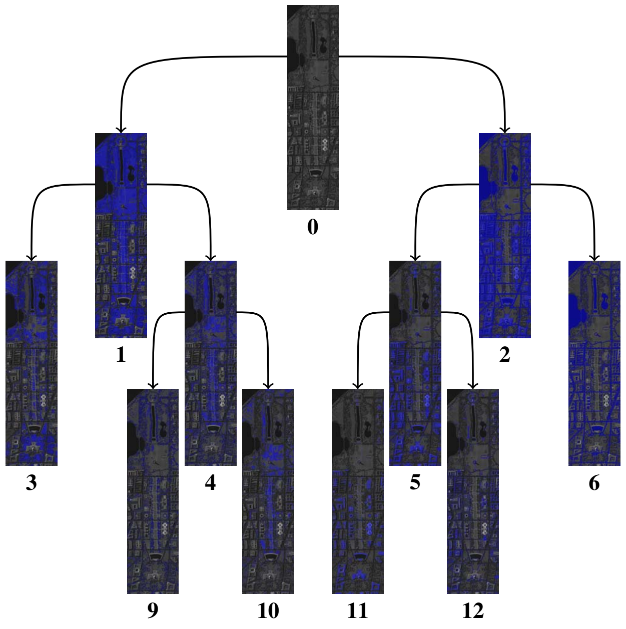


Fig. 1: Hierarchical Clustering of DC Mall HSI

sometimes superlinear dependence of flat NMF algorithms.

We demonstrate in Section IV the efficiency and scalability of our parallel algorithm on three data sets, including the HSI data of the DC mall and an image classification data set involving skin melanoma. The experimental results show that our parallelization of Rank-2 NMF is highly scalable, maintaining computation bound performance on 1000s of cores. We also show the limits of strong scalability when scaling to large numbers of clusters (leaf nodes), as the execution time shifts to becoming interprocessor bandwidth bound and eventually latency bound. The image classification data set requires 800 GB of memory across multiple nodes to process, and in scaling from 10 nodes to 80 nodes of the Summit supercomputer (see § IV-A), we demonstrate parallel speedups of 7.1× for a single Rank-2 NMF and 5.9× for a complete hierarchical clustering.

II. Preliminaries and Related Work

A. *Non-negative Matrix Factorization(NMF)*

The NMF constrained optimization problem

$$\min_{\mathbf{W},\mathbf{H}\geqslant\mathbf{0}} \|\mathbf{A}-\mathbf{W}\mathbf{H}^{\mathsf{T}}\|_2$$

is nonlinear and nonconvex, and various optimization techniques can be used to approximately solve it. A popular approach is to use alternating optimization of the two factor matrices because each subproblem is a nonnegative least squares (NNLS) problem, which is convex and can be solved exactly. Many block coordinate descent (BCD) approaches are possible [13], and one 2-block BCD algorithm that solves the NNLS subproblems exactly is block principal pivoting [14]. This NNLS algorithm is an active-set-like method that determines the sets of entries in the solution vectors that are zero and those that are positive through an iterative but finite process.

When the rank of the factorization (the number of columns of $\mathbf{W}$ and $\mathbf{H}$) is 2, the NNLS subproblems can be solved much more quickly because the number of possible active sets is only 4. As explained in more detail in § III-A1, the optimal solution across the 4 sets can be determined efficiently to solve the NNLS subproblem more quickly than general-rank approaches like block principal pivoting. Because of the relative ease of solving the NMF problem for the rank-2 case, Kuang and Park [15] propose a recursive method to use a rank-2 NMF to partition the input data into 2 parts, whereby each part can be further partitioned via rank-2 NMF of the corresponding original data. This approach yields a hierarchical factorization, potentially uncovering more global structure of the input data and allowing for better scalability of the algorithm to large NMF ranks.

The hierarchical rank-2 NMF method has been applied to document clustering [15] and hyperspectral image segmentation [9]. The leaves of the tree also yield a set of column vectors that can be aggregated into an approximate $\mathbf{W}$ factor (ignoring their hierarchical structure). Using this factor matrix to initialize a higher-rank NMF computation leads to quick convergence and overall faster performance than initializing NMF with random data; this approach is known as Divide-and-Conquer NMF [6]. We focus in this paper on parallelizing the hierarchical algorithms proposed by Kuang and Park [15] and Gillis et al. [9].

B. *Parallel NMF*

Scaling algorithms for NMF to large data often requires parallelization in order to fit the data across the memories of multiple compute nodes or speed up the computation to complete in reasonable time. Parallelizations of multiple optimization approaches have been proposed for general NMF [2], [5], [8], [12], [17]. In particular, we build upon the work of Kannan et al. [7], [11], [12] and the open-source library PLANC, designed for nonnegative matrix and tensor factorizations of dense and sparse data. In this parallelization, the alternating optimization approach is employed with various options for the algorithm used to (approximately) solve the NNLS subproblems. The efficiency of the parallelization is based on scalable algorithms for the parallel matrix multiplications involved in all NNLS algorithms; these algorithms are based on Cartesian distributions of the input matrix across 1D or 2D processor grids.

C. *Communication Model*

We use the α-β-γ model [1], [3], [20] for analysis of distributed-memory parallel algorithms. In this model, the cost of sending a single message of n words of data between two processors is $\alpha+\beta\cdot n$, so that α represents the latency cost of the message and β represents the bandwidth cost of each word in the message. The γ parameter represents the computational cost of a single floating point operation (flop). In this simplified communication model, we ignore contention in the network, assuming in effect a fully connected network, and other limiting factors in practice such as the number of hops between nodes and the network injection rate [10]. We let p represent the number of processors available on the machine.

All of the interprocessor communication in the algorithms presented in this work are encapsulated in collective communication operations that involve the full set of processors. Algorithms for implementing the collective operations are built out of pairwise send and receive operations, and we assume the most efficient algorithms are used in our analysis [3], [20]. The collectives used in our algorithms are all-reduce, all-gather, and reduce-scatter. In an all-reduce, all processors start out with the same amount of data and all end with a copy of the same result, which is in our case a sum of all the inputs (and the same size as a single input). The cost of an all-reduce of size n words is $\alpha \cdot O(\log p) + (\beta + \gamma) \cdot O(n)$ for $n > p$ and $\alpha \cdot O(\log p) + (\beta + \gamma) \cdot O(n \log p)$ for $n < p$. In an all-gather, all processors start out with separate data and all end with a copy of the same result, which is the union of all the input data. If each processor starts with n/p data and ends with n data, the cost of the all-gather is $\alpha \cdot O(\log p) + \beta \cdot O(n)$. In a reduce-scatter, all processors start out with the same amount of data and all end with a subset of the result, which is in our case a sum of all the inputs (and is smaller than its input). If each processor starts with n data and ends with n/p data, the cost of the reduce-scatter is $\alpha \cdot O(\log p) + (\beta + \gamma) \cdot O(n)$. In the case of all-reduce and reduce-scatter, the computational cost is typically dominated by the bandwidth cost because $\beta \gg \gamma$.

III. Algorithms

A. Sequential Algorithms

1) Rank-2 NMF: Using the 2-block BCD approach for a rank-2 NMF yields NNLS subproblems of the form $\min_{\bar{\mathbf{H}} \geqslant \mathbf{0}} \|\mathbf{W}\bar{\mathbf{H}}^{\mathsf{T}} - \mathbf{A}\|$ and $\min_{\bar{\mathbf{W}} \geqslant \mathbf{0}} \|\mathbf{H}\bar{\mathbf{W}}^{\mathsf{T}} - \mathbf{A}^{\mathsf{T}}\|$. In each case, the columns of the transposed variable matrix can be computed independently. Considering the ith row of $\bar{\mathbf{H}}$, for example, the NNLS problem to solve is

$$\min_{\bar{h}_{i,1}, \bar{h}_{i,2} \geqslant 0} \left\| \begin{bmatrix} \mathbf{w}_1 & \mathbf{w}_2 \end{bmatrix} \begin{bmatrix} \bar{h}_{i,1} \\ \bar{h}_{i,2} \end{bmatrix} - \mathbf{a}_i \right\| = \min_{\bar{h}_{i,1}, \bar{h}_{i,2} \geqslant 0} \left\| \bar{h}_{i,1}\mathbf{w}_1 + \bar{h}_{i,2}\mathbf{w}_2 - \mathbf{a}_i \right\|$$

where $\mathbf{w}_1$ and $\mathbf{w}_2$ are the two columns of $\mathbf{W}$ and $\mathbf{a}_i$ is the i column of $\mathbf{A}$. We note that there are four possibilities of solutions, as each of the two variables may be positive or zero.

As shown by Kuang and Park [15], determining which of the four possible solutions is feasible and optimal can be done efficiently by exploiting the following properties:

- if the solution to the unconstrained least squares problem admits two positive values, it is the optimal solution to the nonnegatively constrained problem,
- if $\mathbf{W}$ and $\mathbf{A}$ are both nonnegative, then the candidate solution with two zero values is never (uniquely) optimal and can be discarded, and
- if the unconstrained problem does not admit a positive solution, the better of the two remaining solutions can be determined by comparing $\mathbf{a}_j^{\mathsf{T}}\mathbf{w}_1/\|\mathbf{w}_1\|$ and $\mathbf{a}_j^{\mathsf{T}}\mathbf{w}_2/\|\mathbf{w}_2\|$.

If the unconstrained problem is solved via the normal equations, then the temporary matrices computed for the normal equations ($\mathbf{W}^{\mathsf{T}}\mathbf{W}$ and $\mathbf{A}^{\mathsf{T}}\mathbf{W}$) can be re-used to determine the better of the two solutions with a single positive variable.

Algorithm 1 implements this strategy for all rows of $\mathbf{H}$ simultaneously. It takes as input the matrices $\mathbf{C} = \mathbf{A}^{\mathsf{T}}\mathbf{W}$ and $\mathbf{G} = \mathbf{W}^{\mathsf{T}}\mathbf{W}$, first solves the normal equations for the unconstrained problem, and then chooses between the two alternate possibilities as necessary. We note that each row of $\mathbf{H}$ is independent, and therefore this algorithm is easily parallelized. Solving for $\mathbf{W}$ can be done using inputs $\mathbf{C} = \mathbf{A}\mathbf{H}$ and $\mathbf{G} = \mathbf{H}^{\mathsf{T}}\mathbf{H}$.

Algorithm 1 Rank-2 Nonnegative Least Squares Solve [15]

```
Require: $\mathbf{C}$ is $n \times 2$ and $\mathbf{G}$ is $2 \times 2$ and s.p.d.
 1: function $\mathbf{H} =$ RANK2-NLS-SOLVE($\mathbf{C}, \mathbf{G}$)
 2:     $\mathbf{H} = \mathbf{C}\mathbf{G}^{-1}$                % Solve unconstrained system
 3:     for $i = 1$ to $n$ do
 4:         if $h_{i1} < 0$ or $h_{i2} < 0$ then
 5:             % Choose between single-variable solutions
 6:             if $c_{i1}/\sqrt{g_{11}} < c_{i2}/\sqrt{g_{22}}$ then
 7:                 $h_{i1} = 0$
 8:                 $h_{i2} = c_{i2}/g_{22}$
 9:             else
10:                 $h_{i1} = c_{i1}/g_{11}$
11:                 $h_{i2} = 0$
12:             end if
13:         end if
14:     end for
15: end function
Ensure: $\mathbf{H} = \arg\min_{\bar{\mathbf{H}} \geqslant \mathbf{0}} \|\mathbf{A} - \mathbf{W}\bar{\mathbf{H}}^{\mathsf{T}}\|$ is $n \times 2$ with $\mathbf{C} = \mathbf{A}^{\mathsf{T}}\mathbf{W}$
    and $\mathbf{G} = \mathbf{W}^{\mathsf{T}}\mathbf{W}$
```

Given that the computational complexity of Algorithm 1 is $O(n)$ (or $O(m)$ when computing $\mathbf{W}$), and the complexity of computing $\mathbf{W}^{\mathsf{T}}\mathbf{W}$ and $\mathbf{H}^{\mathsf{T}}\mathbf{H}$ is $O(m + n)$, the typical dominant cost of each iteration of Rank-2 NMF is that of computing $\mathbf{A}^{\mathsf{T}}\mathbf{W}$ and $\mathbf{A}\mathbf{H}$, which is $O(mn)$.

2) Hierarchical Clustering: A Rank-2 NMF can be used to partition the columns of the matrix into two parts. In this case, the columns of the $\mathbf{W}$ factor represent feature weights for each of the two latent components, and the strength of membership in the two components for each column of $\mathbf{A}$ is given by the two values in the corresponding row of $\mathbf{H}$. We can determine part membership by comparing those values: if $h_{i1} > h_{i2}$, then column i of $\mathbf{A}$ is assigned to the first part, which is associated with feature vector $\mathbf{w}_1$. Membership can be determined by other metrics that also take into account balance across parts or attempt to detect outliers.

Given Rank-2 NMF as a splitting procedure, hierarchical clustering builds a binary tree such that each node corresponds to a subset of samples from the original data set and each node's children correspond to a 2-way partition of the node's samples. In this way, the leaves form a partition of the original data, and the internal nodes specify the hierarchical relationship among clusters. As the tree is built, nodes are

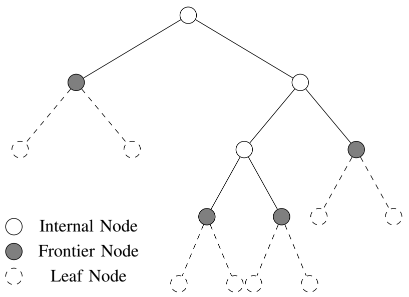


Fig. 2: Hierarchy node classification

split in order of their score, or relative value to the overall clustering of the data. The process can be continued until a target number of leaves is produced or until all remaining leaves have a score below a given threshold.

A node's score can be computed in different ways. For document clustering, Kuang and Park [15] propose using modified normalized discounted cumulative gain, which measures how distinct a node's children are from each other using the feature weights associated with the node and its children. For hyperspectral imaging data, Gillis et al. [9] propose using the possible reduction in overall NMF error if the node is split – the difference in error between using the node itself or using its children. We use the latter in our implementation.

In any case, a node's score depends on properties of its children, so the computation for a split must be done before the split is actually accepted. To this end, we define a *frontier* node to be a parent of leaves; these are nodes whose children have been computed but whose splits have not been accepted. Figure 2 depicts the classification of nodes into internal, frontier, and leaf nodes. As the tree is built, the algorithm selects the frontier node with the highest score to split, though no computation is required to split the node. When a frontier node split is accepted, it becomes an internal node and its children are split (so that their scores can be computed) and added to the set of frontier nodes. When the algorithm terminates, the leaves are discarded and the frontier nodes become the leaves of the output tree.

Our hierarchical clustering algorithm is presented in Algorithm 2 and follows that of Kuang and Park [15]. Each node includes a field $\mathbf{A}$, which is a subset of columns (samples) of the original data, a feature vector $\mathbf{w}$, which is its corresponding column of the $\mathbf{W}$ matrix from its parent's Rank-2 NMF, a score, and pointers to its left and right children. A priority queue $\mathcal{Q}$ tracks the frontier nodes so that the node with the highest score is split at each step of the algorithm. We use a target number of leaf clusters k as the termination condition. When a node is selected from the priority queue, it is removed from the set of frontier nodes and its children are added.

The splitting procedure is specified in Algorithm 3. After the Rank-2 NMF is performed, the $\mathbf{H}$ factor is used to determine part membership, and the columns of the $\mathbf{W}$ factor are assigned to the child nodes. The score of the node is computed as the reduction in overall NMF error if the node is split, which can be computed from the principal singular values of the subsets of columns of the node and its children, as given in Line 6. The principal singular values of the children are computed via the power method. Note that the principal singular value of the node itself need not be recomputed as it was needed for its parent's score.

Algorithm 2 Hierarchical Clustering [15]

```
Require: $\mathbf{A}$ is $m \times n$, $k$ is target number of leaf clusters
 1: function $\mathcal{T}$ = HIER-R2-NMF($\mathbf{A}$)
 2:     $\mathcal{R}$ = node($\mathbf{A}$)                 % create root node
 3:     SPLIT($\mathcal{R}$)
 4:     inject($\mathcal{Q}$,$\mathcal{R}$.left)           % create priority queue
 5:     inject($\mathcal{Q}$,$\mathcal{R}$.right)          % of frontier nodes
 6:     while size($\mathcal{Q}$) < $k$ do
 7:         $\mathcal{N}$ = eject($\mathcal{Q}$)       % frontier node with max score
 8:         SPLIT($\mathcal{N}$.left)              % split left child
 9:         inject($\mathcal{Q}$,$\mathcal{N}$.left)       % and add to $\mathcal{Q}$
10:         SPLIT($\mathcal{N}$.right)             % split right child
11:         inject($\mathcal{Q}$,$\mathcal{N}$.right)      % and add to $\mathcal{Q}$
12:     end while
13: end function
Ensure: $\mathcal{T}$ is binary tree rooted at $\mathcal{R}$ with $k$ frontier nodes,
    each node has subset of cols of $\mathbf{A}$ and feature vector $\mathbf{w}$
```

Algorithm 3 Node Splitting via Rank-Two NMF

```
Require: $\mathcal{N}$ has a subset of columns given by field $\mathbf{A}$
 1: function SPLIT($\mathcal{N}$)
 2:     [$\mathbf{W}$,$\mathbf{H}$] = RANK2-NMF($\mathcal{N}$.$\mathbf{A}$)          % split $\mathcal{N}$
 3:     partition $\mathcal{N}$.$\mathbf{A}$ into $\mathbf{A}_1$ and $\mathbf{A}_2$ using $\mathbf{H}$
 4:     $\mathcal{N}$.left = node($\mathbf{A}_1$,$\mathbf{w}_1$)          % create left child
 5:     $\mathcal{N}$.right = node($\mathbf{A}_2$,$\mathbf{w}_2$)         % create right child
 6:     $\mathcal{N}$.score = $\sigma_1^2(\mathbf{A}_1)+\sigma_1^2(\mathbf{A}_2)-\sigma_1^2(\mathcal{N}.\mathbf{A})$
 7: end function
Ensure: $\mathcal{N}$ has two children and a score
```

B. Parallelization

In this section, we consider the options for parallelizing Hierarchical Rank-2 NMF Clustering (Algorithm 2) and provide an analysis for our approach. The running time of an algorithm is data dependent because not only does each Rank-2 NMF computation require a variable number of iterations, but also the shape of the tree can vary from a balanced binary tree with $O(\log k)$ levels to a tall, flat tree with $O(k)$ levels. For the sake of analysis, we will assume a fixed number of NMF iterations for every node of the tree and we will analyze the cost of complete levels.

The first possibility for parallelization is across the nodes of the tree, as each Rank-2 NMF split is independent. We choose not to parallelize across nodes in the tree for two reasons. The first reason is that while the NMF computations are independent, choosing which nodes to split may depend on global information. In particular, when the global target is to determine k leaf clusters, the nodes must be split in order of their scores, which leads to a serialization of the node splits. This serialization might be relaxed using speculative execution, but it risks performing unnecessary computation. If the global target is to split all nodes with sufficiently high scores, then this serialization is also avoided and node splits become truly independent. We choose not to parallelize in this way to remain agnostic to the global stopping criterion.

The second reason is that parallelizing across nodes requires redistribution of the input data. Given a node split by $\hat{p}$ processors, in order to assign disjoint sets of processors to each child node, each of the $\hat{p}$ processors would have to redistribute their local data, sending data for samples not in their child's set and receiving data for those in their child's set. The communication would be data dependent, but on average, each processor would communicate half of its data in the redistribution set, which could have an all-to-all communication pattern among the $\hat{p}$ processors. For a node with $\hat{n}$ columns, the communication cost would be at least $O(m\hat{n}/\hat{p})$ words, which is much larger than the communication cost per iteration of Parallel Rank-2 NMF, as we will see in § III-B2.

By choosing not to parallelize across nodes in the tree, we employ all p processors on each node, and split nodes in sequence. The primary computations used to split a node are the Rank-2 NMF and the score computation, which is based on an approximation of the largest singular value. We use an alternating-updating algorithm for Rank-2 NMF as described in Section II, and we parallelize it following the methodology proposed in [7] and presented in Algorithm 4.

The communication cost of the algorithm depends on the parallel distribution of the input matrix data $\mathbf{A}$. In order to avoid redistribution of the matrix data, we choose a 1D row distribution so that each processor owns a subset of the rows of $\mathbf{A}$. Because the clustering partition splits the columns of $\mathbf{A}$, each processor can partition its local data into left and right children to perform the split without any interprocessor communication. If we use a 2D distribution for a given node, then because the partition is data dependent, a data redistribution is required in order to obtain a load balanced distribution of both children. Figure 3 presents a visualization of the node-splitting process using a 1D processor distribution. In the following subsections, we describe the parallel algorithms for Rank-2 NMF and approximating the principal singular value given this 1D data distribution and analyze their complexity in the context of the hierarchical clustering algorithm.

1) Algorithms:

a) Parallel Rank-2 NMF*:* Algorithm 4 presents the parallelization of an alternating-updating scheme for NMF that uses the exact rank-2 solve algorithm presented in Algorithm 1 to update each factor matrix. The algorithm computes the inputs to the rank-2 solves in parallel and then exploits the parallelism across rows of the factor matrix so that each processor solves for a subset of rows simultaneously. The distribution of all matrices is 1D row distribution, so that each processor owns a subset of the rows of $\mathbf{A}$, $\mathbf{W}$, and $\mathbf{H}$. We use the notation $\hat{\mathbf{A}}$ to refer to the $(m/p) \times n$ local data matrix and $\hat{\mathbf{W}}$ and $\hat{\mathbf{H}}$ to refer to the $(m/p) \times 2$ and $(n/p) \times 2$ local factor matrices. With this distribution, the computation of $\mathbf{W}^\mathsf{T}\mathbf{W}$ and $\mathbf{H}^\mathsf{T}\mathbf{H}$ each is done via local multiplication followed by a single all-reduce collective. All processors own the data they need to

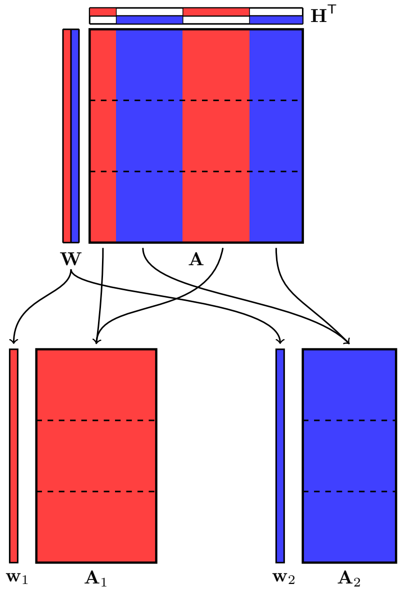


Fig. 3: Parallel splitting using Rank-2 NMF and 1D processor distribution. A Rank-2 NMF computes factor matrices $\mathbf{W}$ and $\mathbf{H}$ to approximate $\mathbf{A}$, the values of $\mathbf{H}$ are used to determine child membership of each column (either red or blue), and the corresponding column of the $\mathbf{W}$ matrix represents the part's feature weighting. The 1D distribution is depicted for 3 processors to show that splitting requires no interprocessor redistribution as children are evenly distributed identically to the parent.

compute their contribution to $\mathbf{A}^\mathsf{T}\mathbf{W}$; in order to distribute the result to compute the rows $\mathbf{H}$ independently, a reduce-scatter collective is used to sum and simultaneously distribute across processors. To obtain the data needed to compute $\hat{\mathbf{W}}$, each processor must access all of $\mathbf{H}$, which is performed via an all-gather collective. The iteration progresses until a convergence criterion is satisfied. For performance benchmarking we use a fixed number of iterations, and in practice we use relative change in objective function value (residual norm).

b) Parallel Power Method*:* In order to compute the score for a frontier node, we use the difference between the principal singular value of the matrix columns of the node and the sum of those of its children. Thus, we must determine the principal singular value of every node in the tree once, including leaf nodes. We use the power method to approximate it, repeatedly applying $\mathbf{A}\mathbf{A}^\mathsf{T}$ to a vector until it converges to the leading right singular vector. We present the power method in Algorithm 5. Note that we do not normalize the approximate left singular vector so that the computed value approximates the square of the largest singular value.

Given the 1D distribution, only one communication collective is required for the pair of matrix-vector multiplications. That is, the approximate right singular vector $\mathbf{v}$ is redundantly owned on each processor, and the approximate left singular vector $\mathbf{u}$ is distributed across processors. Each processor can compute its local $\hat{\mathbf{u}}$ from $\mathbf{v}$ without communication and use

Algorithm 4 Parallel Rank-2 NMF

```
Require: $\mathbf{A}$ is $m \times n$ and row-distributed across processors
    so that $\hat{\mathbf{A}}$ is local $(m/p) \times n$ submatrix
1: function $[\mathbf{W},\mathbf{H}] =$ PARALLEL-RANK2-NMF($\mathbf{A}$)
2:     Initialize local $\hat{\mathbf{W}}$ randomly
3:     while not converged do
4:         % Compute $\mathbf{H}$
5:         $\hat{\mathbf{G}}_W = \hat{\mathbf{W}}^\mathsf{T}\hat{\mathbf{W}}$
6:         $\mathbf{G}_W =$ ALL-REDUCE($\hat{\mathbf{G}}_W$)
7:         $\hat{\mathbf{B}} = \hat{\mathbf{A}}^\mathsf{T}\hat{\mathbf{W}}$
8:         $\hat{\mathbf{C}} =$ REDUCE-SCATTER($\hat{\mathbf{B}}$)
9:         $\hat{\mathbf{H}} =$ RANK2-NLS-SOLVE($\hat{\mathbf{C}}$,$\mathbf{G}_W$)
10:        % Compute $\mathbf{W}$
11:        $\hat{\mathbf{G}}_H = \hat{\mathbf{H}}^\mathsf{T}\hat{\mathbf{H}}$
12:        $\mathbf{G}_H =$ ALL-REDUCE($\hat{\mathbf{G}}_H$)
13:        $\mathbf{H} =$ ALL-GATHER($\hat{\mathbf{H}}$)
14:        $\hat{\mathbf{D}} = \hat{\mathbf{A}}\mathbf{H}$
15:        $\hat{\mathbf{W}} =$ RANK2-NLS-SOLVE($\hat{\mathbf{D}}$,$\mathbf{G}_H$)
16:    end while
17: end function
Ensure: $\mathbf{A} \approx \mathbf{W}\mathbf{H}^\mathsf{T}$ with $\mathbf{W}$, $\mathbf{H}$ row-distributed
```

the result for its contribution to $\mathbf{v} = \mathbf{A}^\mathsf{T}\mathbf{u}$. An all-reduce collective is used to obtain a copy of $\mathbf{v}$ on every processor for the next iteration, and the norm is redundantly computed without further communication. We used the relative change in σ as the stopping criterion for benchmarking.

Algorithm 5 Parallel Power Method

```
Require: $\mathbf{A}$ is $m \times n$ and row-distributed across processors
    so that $\hat{\mathbf{A}}$ is local $(m/p) \times n$ submatrix
1: function $\sigma =$ PARALLEL-POWER-METHOD($\mathbf{A}$)
2:     Initialize $\mathbf{v}$ randomly and redundantly
3:     while not converged do
4:         $\hat{\mathbf{u}} = \hat{\mathbf{A}}\mathbf{v}$
5:         $\hat{\mathbf{z}} = \hat{\mathbf{A}}^\mathsf{T}\hat{\mathbf{u}}$
6:         $\mathbf{v} =$ ALL-REDUCE($\hat{\mathbf{z}}$)
7:         $\sigma = \|\mathbf{v}\|$
8:         $\mathbf{v} = \mathbf{v}/\sigma$
9:     end while
10: end function
Ensure: $\sigma \approx \sigma_1^2(\mathbf{A})$ is redundantly owned by all procs
```

2) Analysis:

a) Parallel Rank-2 NMF*:* Each iteration of Algorithm 4 incurs the same cost, so we analyze per-iteration computation and communication costs. We first consider the cost of the Rank-2 NNLS solves, which are local computations. In the notation of Algorithm 1, matrix $\mathbf{G}$ is 2×2, so solving the unconstrained system (via Cholesky decomposition) and then choosing between single-positive-variables solutions if necessary requires constant time per row of $\mathbf{C}$. Thus, the cost of Algorithm 1 is proportional to the number of rows of the first input matrix. In the context of Algorithm 4, the per-iteration computational cost of rank-2 solves is then $O((m+n)/p)$. The other local computations are the matrix multiplications $\hat{\mathbf{W}}^\mathsf{T}\hat{\mathbf{W}}$ and $\hat{\mathbf{H}}^\mathsf{T}\hat{\mathbf{H}}$, which also amount to $O((m+n)/p)$ flops, and $\hat{\mathbf{A}}^\mathsf{T}\hat{\mathbf{W}}$ and $\hat{\mathbf{A}}\mathbf{H}$, which require $O(mn/p)$ flops because they involve the data matrix. Thus, the computation cost is $\gamma \cdot O((mn+m+n)/p)$ and typically dominated by the multiplications involving $\mathbf{A}$. We track the lower order terms corresponding to NNLS solves because their hidden constants are larger than that of the dominating term.

There are four communication collectives each iteration, and each involves all p processors. The two all-reduce collectives to compute the Gram matrices of the factor matrices involve 2×2 matrices and incur a communication cost of $(\gamma+\beta+\alpha) \cdot O(\log p)$. The reduce-scatter and all-gather collectives involve $n \times 2$ matrices (the size of $\mathbf{H}$) and require $\beta \cdot O(n) + \alpha \cdot O(\log p)$ in communication cost (we ignore the computation cost of the reduce-scatter because it is typically dominated by the bandwidth cost). If the algorithm performs $\imath$ iterations, the overall cost of Algorithm 4 is

$$\gamma \cdot O\left(\frac{\imath(mn+m+n)}{p}\right) + \beta \cdot O(\imath n) + \alpha \cdot O(\imath \log p). \quad (1)$$

b) Parallel Power Method*:* Similar to the previous analysis, we consider a single iteration of the power method. The local computation is dominated by two matrix-vector products involving the local data matrix of size $O(mn/p)$ words, incurring $O(mn/p)$ flops. The single communication collective is an all-reduce of the approximate right singular vector, which is of size n, incurring $\beta \cdot O(n) + \alpha \cdot O(\log p)$ communication. We ignore the $O(n)$ computation cost of normalizing the vector, as it will typically be dominated by the communication cost of the all-reduce. Over $\jmath$ iterations, Algorithm 5 has an overall cost of

$$\gamma \cdot O\left(\frac{\jmath mn}{p}\right) + \beta \cdot O(\jmath n) + \alpha \cdot O(\jmath \log p). \quad (2)$$

Note the per-iteration cost of the power method differs by only a constant from the per-iteration cost of Rank-2 NMF. Because the power method involves single vectors rather than factor matrices with two columns, its constants are smaller than half the size of their counterparts.

c) Hierarchical Clustering*:* To analyze the overall cost of the hierarchical clustering algorithm, we sum the costs over all nodes in the tree. Because the shape of the tree is data dependent and affects the overall costs, for the sake of analysis we will analyze only complete levels. The number of rows in any node is m, the same as the root node, as each splitting corresponds to a partition of the columns. Furthermore, because each split is a partition, every column of $\mathbf{A}$ is represented exactly once in every complete level of the tree. If we assume that all nodes perform the same number of NMF iterations ($\imath$) and power method iterations ($\jmath$), then the dominating costs of a node with $\overline{n}$ columns is

$$\gamma \cdot O\left(\frac{(\imath+\jmath)m\overline{n} + \imath(m+\overline{n})}{p}\right) + \beta \cdot O((\imath+\jmath)\overline{n}) + \alpha \cdot O((\imath+\jmath)\log p).$$

Because the sum of the number of columns across any level of the tree is n, the cost of the ℓth level of the tree is

$$\gamma \cdot O\left(\frac{(\imath+\jmath)mn+\imath m2^{\ell}}{p}\right)+\beta \cdot O((\imath+\jmath)n) \\ +\alpha \cdot O((\imath+\jmath)2^{\ell}\log p). \quad (3)$$

Note that the only costs that depend on the level index ℓ are the latency cost and a lower-order computational cost.

Summing over levels and assuming the tree is nearly balanced and has height $O(\log k)$ where k is the number of frontier nodes, we obtain an overall cost of Algorithm 2 of

$$\gamma \cdot O\left(\frac{(\imath+\jmath)mn}{p}\log k+\frac{\imath mk}{p}\right)+\beta \cdot O((\imath+\jmath)n\log k) \\ +\alpha \cdot O((\imath+\jmath)k\log p). \quad (4)$$

We see that the leading order computational cost is logarithmic in k and perfectly load balanced. If the overall running time is dominated by the computation (and in particular the matrix multiplications involving $\mathbf{A}$), we expect near-perfect strong scaling. The bandwidth cost is also logarithmic in k but does not scale with the number of processors. The latency cost grows most quickly with the target number of clusters k but is also independent of the matrix dimensions m and n.

IV. Experimental Results

A. Experimental Platform

All the experiments in this section were conducted on Summit. Summit is a supercomputer created by IBM for the Oak Ridge National Laboratory. There are approximately 4,600 nodes on Summit. Each node contains two IBM POWER9 processors on separate sockets with 512 GB of DDR4 memory. Each POWER9 processor utilizes 22 IBM SIMD Multi-Cores (SMCs), although one of these SMCs on each processor is dedicated to memory transfer and is therefore not available for computation. For node scaling experiments, all 42 available SMCs were utilized in each node so that every node computed with 42 separate MPI processes. Additionally, every node also supports six NVIDIA Volta V100 accelerators but these were unused by our algorithm.

Our implementation builds on the PLANC open-source library [7] and uses the Armadillo library (version 9.900.1) for all matrix operations. On Summit, we linked this version of Armadillo with OpenBLAS (version 0.3.9) and IBM's Spectrum MPI (version 10.3.1.2-20200121).

B. Datasets

a) Hyperspectral Imaging: We use the Hyperspectral Digital Imagery Collection Experiment (HYDICE) image of the Washington DC Mall. We will refer to this dataset as DC-HYDICE [16]. DC-HYDICE is formatted into a 3-way tensor representing two spatial dimensions of pixels and one dimension of spectral bands. So, a slice along the spectral band dimension would be the full DC-HYDICE image in that spectral band. For hierarchical clustering, these tensors are flattened so that the rows represent the 191 spectral bands and the columns represent the 392960 pixels. The data set is approximately 600 MB in size.

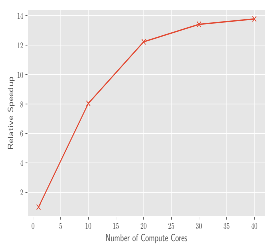


Fig. 4: Strong Scaling for Clustering on DC-HYDICE

b) Image Classification: The SIIM-ISIC Melanoma classification dataset, which we will refer to as SIIM-ISIC [19], consists of 33126 RGB training images equally sized at 1024×1024. Unlike with hyperspectral imaging, the resulting matrix used in hierarchical clustering consists of image pixels along the rows and individual images along the columns. So, the resulting sized matrix is 3145728×33126, which is approximately 800 GB in size. Given its size, SIIM-ISIC requries 10 Summit nodes to perform hierarchical clustering.

c) Synthetic Dataset: Our synthetic dataset has the same aspect ratio of SIIM-ISIC but consists of fewer rows and columns by a factor of 3. The resulting matrix is 1048576×11042. We choose the smaller size in order to fit on a single node for scaling experiments.

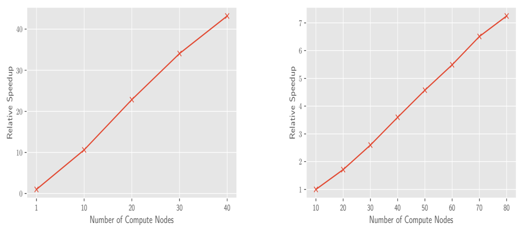
C. Performance

For all hierarchical clustering experiments in this section, the number of tree leaf nodes k was set at 100, the number of NMF iterations was set to 100, the power iteration was allowed to stop iterating after convergence, and only complete levels were considered for analysis purposes for both level and strong scaling plots.

1) Single-Node Scaling for DC Dataset: DC-HYDICE is small compared to the other datasets, so it can easily fit on one compute node. Also, its small number of 191 rows doesn't allow for parallelizing beyond that number of MPI processes. So, this dataset was used for a single-node scaling experiment on Summit from 1 to 42 cores. Because Rank-2 NMF is memory bandwidth bound, we expect limited speedup on one node due to the memory bandwidth not scaling linearly with the number of cores. Figure 4 shows that there is enough speedup (14× on 42 cores) for it to be worth parallelizing such a small problem, but perfect scaling requires more memory bandwidth. In this experiment, the processes were distributed across both sockets so that an even number of cores on each socket are used.

2) Rank-2 NMF Strong Scaling: We perform strong scaling experiments for a single Rank-2 NMF (Algorithm 4) on the synthetic and SIIM-ISIC datasets. The theory (Equation (1)) suggests that perfect strong scaling is possible as long as the execution time is dominated by local computation. Both the matrix multiplications and NNLS solves scale linearly with $1/p$ (we expect MatMul to dominate), but the bandwidth cost is independent of p and latency increases slightly with p.

Figures 5a and 5b show performance relative to the smallest number of compute nodes required to store data and



(a) Synthetic Data (b) SIIM-ISIC Data

Fig. 5: Strong Scaling Speedup for Rank-2 NMF

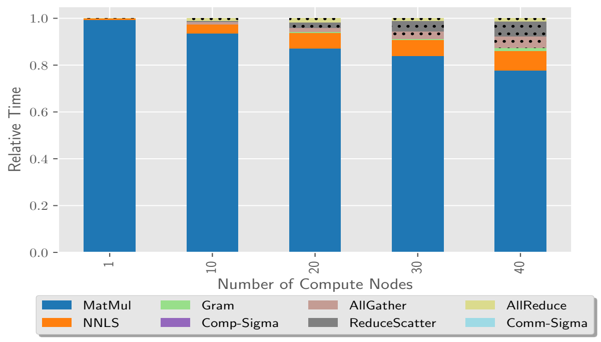


Fig. 6: Time Breakdown for Rank-2 NMF on Synthetic

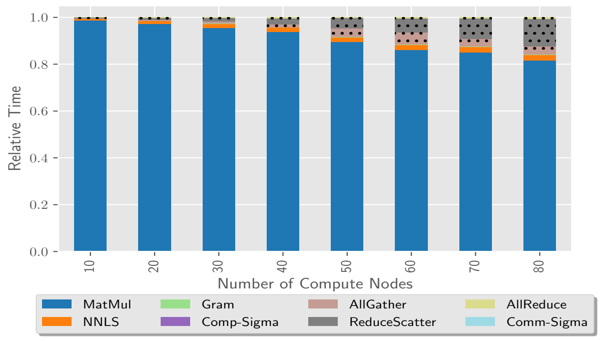
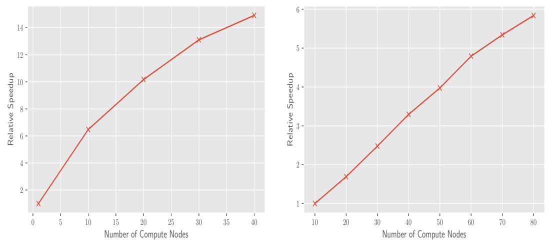


Fig. 7: Time Breakdown for Rank-2 NMF on SIIM-ISIC



(a) Synthetic Data (b) SIIM-ISIC Data

Fig. 8: Strong Scaling Speedup for Clustering

factor matrices. For these data sets, we observe nearly perfect strong scaling, with 42× speedup on 40 compute nodes (over 1 compute node) for synthetic data and 7.1× speedup on 80 compute nodes (over 10 compute nodes) for SIIM-ISIC data.

The relative time breakdowns are presented in Figures 6 and 7 and explain the strong scaling performance. Each experiment is normalized to 100% time, so comparisons cannot be readily made across numbers of compute nodes. For both data sets, we see that the time is dominated by MatMul, which is the primary reason for the scalability. The dominant matrix multiplications are between a large matrix and a matrix with 2 columns, so it is locally memory bandwidth bound, with performance proportional to the size of the large matrix. In each plot, we also see the relative time of all-gather and reduce-scatter increasing, which is because the local computation is decreasing while the communication cost is slightly increasing with p. This pattern will continue as p increases, which will eventually limit scalability, but for these data sets the MatMul takes around 80% of the time at over 2000 cores.

3) Hierarchical Clustering Strong Scaling: From Equation (4), we expect to see perfect strong scaling in a computationally bound clustering problem with target cluster count $k = 100$. As k is large, we expect the latency cost of small problems deep in the tree to limit scalability.

Figure 8a demonstrates the scalability of the synthetic data set on up to 40 nodes, and we observe a 15× speedup compared to 1 node. Figure 9 shows the relative time breakdown and explains the limitation on scaling. On 40 nodes, computation still takes 60% of the total time, but the all-gather and reduce-scatter costs have grown in relative time because they do not scale with p. Because all-reduce involves only a constant amount of data and its time remains relatively small, we conclude the communication is bandwidth bound at this scale.

With the larger SIIM-ISIC dataset, it's possible to scale much further as seen in Figure 8b, where we observe a 5.9× speedup of 80 compute nodes compared to 10. From Figure 10, we see that the communication cost constitutes less than 20% of the total time even at 80 compute nodes.

We note that the speedup of the overall hierarchical clustering algorithm is not as high as for a single Rank-2 NMF (measured at the root node). This is due to inefficiencies in the lower levels of the tree, as we explore in the next section.

4) Level Scaling: To compare execution time across levels of a particular tree, we consider only complete levels. From Equation (3), the dominant computational term (due to MatMul) is constant per level, the lower order computational term (represented by NNLS) grows like $O(2^\ell)$, and the latency cost grows similarly like $O(2^\ell)$.

Figure 11 show absolute time across levels for the synthetic data set on 1 node. The MatMul cost decreases slightly per level, which may be explained by cache effects in the local matrix multiply, as each node's subproblem decreases in size. The NNLS grows exponentially, as expected, and communication is negligible.

Figure 12 shows the level breakdown for the synthetic data on 40 nodes, where we see different behavior. MatMul cost

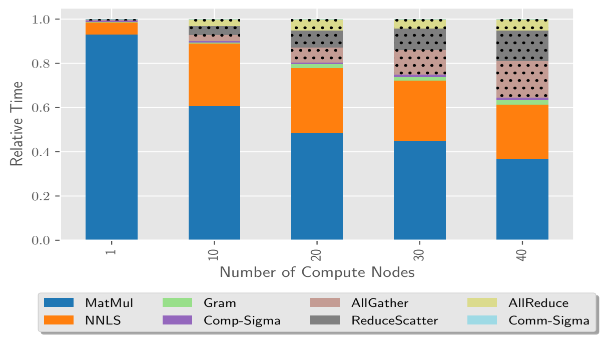


Fig. 9: Time Breakdown for Clustering on Synthetic

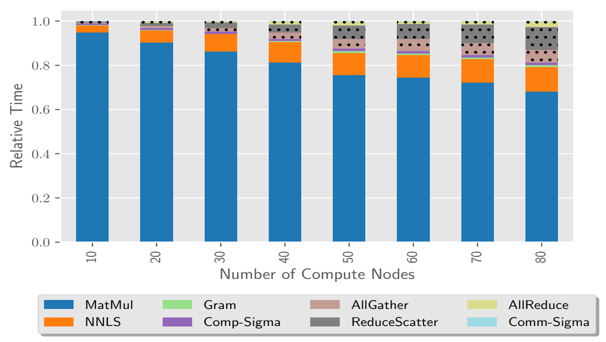


Fig. 10: Time Breakdown for Clustering on SIIM-ISIC

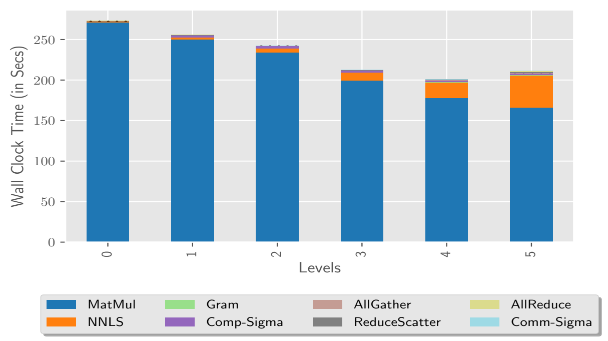


Fig. 11: Level Times for 1 Compute Node on Synthetic

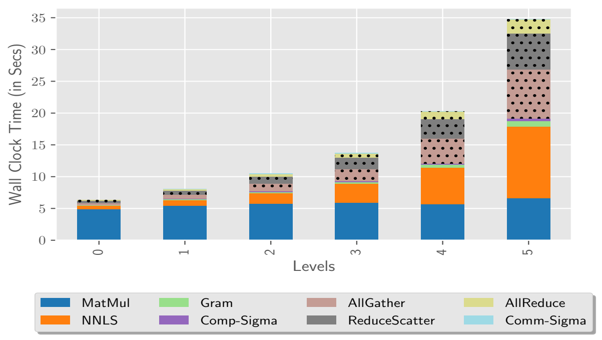
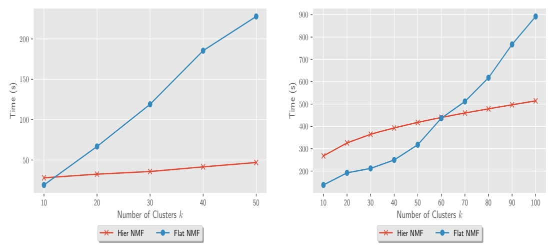


Fig. 12: Level Times for 40 Compute Nodes on Synthetic



(a) DC-HYDICE Data (b) Synthetic Data

Fig. 13: Rank Scaling for Hierarchical and Flat NMF

is again constant across levels and the NNLS cost becomes dominating at lower levels suggesting it does not scale as well as MatMul. We also see all-reduce time becoming significant as communication time increases, indicating that the nodes at lower levels are becoming more latency bound. Thus, we see that the poorer scaling at the lower levels of the tree is the main reason the overall hierarchical clustering algorithm does not scale as well as the single Rank-2 NMF at the root node.

5) Rank Scaling: To confirm the slow growth in running time of the hierarchical algorithm in terms of the number of clusters k, we perform rank scaling experiments for DC-HYDICE and synthetic data. Assuming a balanced tree and relatively small k, Equation (4) shows that the dominant computational cost is proportional to $\log k$, while a flat NMF algorithm has a dominant cost that is linear in k [12]. Figure 13 shows the raw time for various values of k, confirming that running time for HierNMF grows more slowly in k than a flat NMF algorithm (based on Block Principal Pivoting) from PLANC [7] with the same number of columns and processor grid. We see that for sufficiently large k, the hierarchical algorithm outperforms flat NMF and it scales much better with k.

V. Conclusion

As shown in the theoretical analysis (§ III-B2) and experimental results (§ IV-C3), Algorithm 2 can efficiently scale to large p as long as the execution time is dominated by local matrix multiplication. The principal barriers to scalability are the bandwidth cost due to Rank-2 NMF, which is consistent across levels of the tree and proportional to the number of columns n of the original data set, and the latency cost due to large numbers of tree nodes in lower levels of the tree. When n is small relative to m and the number of leaves k and levels ℓ are small, then these barriers do not pose a problem until p is very large. However, if the input matrix is short and fat (i.e., has many samples with few features), then the bandwidth cost can hinder performance for smaller p. Likewise, if k is large or the tree is lopsided, then achieving scalability for very small problems is more difficult. We also note that in the case of sparse $\mathbf{A}$, it becomes more difficult to hide communication behind the cheaper matrix multiplications, and other costs may become more dominant.

One approach for reducing the bandwidth cost of Rank-2 NMF is to choose a more balanced data distribution over a 2D grid, as proposed by Kannan et al. [11]. This will reduce the communicated data and achieve a local data matrix that is more square, which can improve local matrix multiplication performance. The downside of this approach is requiring a redistribution of the data for each split, but if many NMF iterations are required, then the single upfront cost may be amortized.

Another approach to alleviate the rising latency costs of lower levels of the tree is to parallelize across nodes of the tree. This will result in fewer processors working on any given node, reducing the synchronization time among them, and it will allow small, latency-bound problems to be solved simultaneously. Prioritizing the sequence of node splits is more difficult in this case, but modifying the stopping criterion for splitting to use a score threshold instead of a target number of leaves will allow truly independent computation.

In the future, we also plan to compare performance of Algorithm 2 with flat NMF algorithms and employ the Divide-and-Conquer NMF technique [6] of seeding an iterative flat NMF algorithm with the feature vectors of the leaf nodes. The parallel technique proposed here can be combined with the existing PLANC library [7] to obtain faster overall convergence for very large datasets.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. OAC-1642385 and OAC-1642410. This manuscript has been co-authored by UT-Battelle, LLC under Contract No. DE-AC05-00OR22725 with the U.S. Department of Energy. This project was partially funded by the Laboratory Director's Research and Development fund. This research used resources of the Oak Ridge Leadership Computing Facility at the Oak Ridge National Laboratory, which is supported by the Office of Science of the U.S. Department of Energy.

The authors would like to thank Simin Ma for contributions to the algorithmic analysis and John Farrell for his contributions to the implementation of the parallel algorithm.

References

[1] G. Ballard, E. Carson, J. Demmel, M. Hoemmen, N. Knight, and O. Schwartz. Communication lower bounds and optimal algorithms for numerical linear algebra. *Acta Numerica*, 23:1–155, May 2014. doi:10.1017/S0962492914000038.

[2] E. Battenberg and D. Wessel. Accelerating non-negative matrix factorization for audio source separation on multi-core and many-core architectures. In *ISMIR*, pages 501–506, 2009. URL: https://archives.ismir.net/ismir2009/paper/000089.pdf.

[3] E. Chan, M. Heimlich, A. Purkayastha, and R. van de Geijn. Collective communication: theory, practice, and experience. *Concurrency and Computation: Practice and Experience*, 19(13):1749–1783, 2007. doi:10.1002/cpe.1206.

[4] C. Ding, X. He, and H.D. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. In *SDM '05*, pages 606–610. SIAM, 2005. doi:10.1137/1.9781611972757.70.

[5] B. Drake, S. Lee-Urban, and H. Park. Smallk v1.6.2. http://smallk.github.io/, June 2017.

[6] R. Du, D. Kuang, B. Drake, and H. Park. DC-NMF: nonnegative matrix factorization based on divide-and-conquer for fast clustering and topic modeling. *Journal of Global Optimization*, 68(4):777–798, 2017. doi:10.1007/s10898-017-0515-z.

[7] S. Eswar, K. Hayashi, G. Ballard, R. Kannan, M.A. Matheson, and H. Park. PLANC: Parallel low rank approximation with non-negativity constraints. Technical report, 2019. URL: https://arxiv.org/abs/1909.01149.

[8] J.P. Fairbanks, R. Kannan, H. Park, and D.A. Bader. Behavioral clusters in dynamic graphs. *Parallel Computing*, 47:38–50, 2015. doi:10.1016/j.parco.2015.03.002.

[9] N. Gillis, D. Kuang, and H. Park. Hierarchical clustering of hyperspectral images using rank-two nonnegative matrix factorization. *IEEE Transactions on Geoscience and Remote Sensing*, 53(4):2066–2078, April 2015. doi:10.1109/TGRS.2014.2352857.

[10] W. Gropp, L.N. Olson, and P. Samfass. Modeling MPI communication performance on SMP nodes: Is it time to retire the ping pong test. In *EuroMPI '16*, pages 41–50, New York, NY, USA, 2016. ACM. doi:10.1145/2966884.2966919.

[11] R. Kannan, G. Ballard, and H. Park. A high-performance parallel algorithm for nonnegative matrix factorization. In *PPoPP '16*, pages 9:1–9:11, New York, NY, USA, February 2016. ACM. doi:10.1145/2851141.2851152.

[12] R. Kannan, G. Ballard, and H. Park. MPI-FAUN: An MPI-based framework for alternating-updating nonnegative matrix factorization. *IEEE Transactions on Knowledge and Data Engineering*, 30(3):544–558, March 2018. doi:10.1109/TKDE.2017.2767592.

[13] J. Kim, Y. He, and H. Park. Algorithms for nonnegative matrix and tensor factorizations: a unified view based on block coordinate descent framework. *Journal of Global Optimization*, 58(2):285–319, 2014. doi:10.1007/s10898-013-0035-4.

[14] J. Kim and H. Park. Fast nonnegative matrix factorization: An active-set-like method and comparisons. *SIAM Journal on Scientific Computing*, 33(6):3261–3281, 2011. doi:10.1137/110821172.

[15] D. Kuang and H. Park. Fast rank-2 nonnegative matrix factorization for hierarchical document clustering. In *KDD '13*, pages 739–747, New York, NY, USA, 2013. ACM. doi:10.1145/2487575.2487606.

[16] D. Landgrebe and L. Biehl. Multispec - hyperspectral images. https://engineering.purdue.edu/~biehl/MultiSpec/hyperspectral.html, February 2020.

[17] G.E. Moon, J.A. Ellis, A. Sukumaran-Rajam, S. Parthasarathy, and P. Sadayappan. ALO-NMF: Accelerated locality-optimized non-negative matrix factorization. In *KDD '20*, 2020. doi:10.1145/3394486.3403227.

[18] F. Shahnaz, M.W. Berry, V.P. Pauca, and R.J. Plemmons. Document clustering using nonnegative matrix factorization. *Information Processing & Management*, 42(2):373–386, 2006. doi:10.1016/j.ipm.2004.11.005.

[19] The ISIC 2020 challenge dataset, 2020. doi:10.34970/2020-ds01.

[20] R. Thakur, R. Rabenseifner, and W. Gropp. Optimization of collective communication operations in MPICH. *International Journal of High Performance Computing Applications*, 19(1):49–66, 2005. doi:10.1177/1094342005051521.

[21] W. Xu, X. Liu, and Y. Gong. Document clustering based on non-negative matrix factorization. In *SIGIR '03*, pages 267–273, 2003. doi:10.1145/860435.860485.