Memory-efficient Parallel Tensor Decompositions

Muthu Baskaran, Tom Henretty, Benoit Pradelle,
M. Harper Langston, David Bruns-Smith, James Ezick, Richard Lethin
Reservoir Labs Inc.
New York, NY 10012
Email: {baskaran,henretty,pradelle,langston,bruns-smith,ezick,lethin}@reservoir.com

***Abstract*—Tensor decompositions are a powerful technique for enabling comprehensive and complete analysis of real-world data. Data analysis through tensor decompositions involves intensive computations over large-scale irregular sparse data. Optimizing the execution of such data intensive computations is key to reducing the time-to-solution (or response time) in real-world data analysis applications. As high-performance computing (HPC) systems are increasingly used for data analysis applications, it is becoming increasingly important to optimize sparse tensor computations and execute them efficiently on modern and advanced HPC systems. In addition to utilizing the large processing capability of HPC systems, it is crucial to improve memory performance (memory usage, communication, synchronization, memory reuse, and data locality) in HPC systems. In this paper, we present multiple optimizations that are targeted towards faster and memory-efficient execution of large-scale tensor analysis on HPC systems. We demonstrate that our techniques achieve reduction in memory usage and execution time of tensor decomposition methods when they are applied on multiple datasets of varied size and structure from different application domains. We achieve up to 11x reduction in memory usage and up to 7x improvement in performance. More importantly, we enable the application of large tensor decompositions on some important datasets on a multi-core system that would not have been feasible without our optimization.**

I. Introduction

Tensors, or multi-dimensional arrays, are a natural way of representing multi-aspect data. Tensor decompositions are increasingly becoming prominent as a powerful technique for extracting and explaining the semantic properties of data in real-world data analysis applications. Tensor decompositions have applications in a range of domains such as cybersecurity, geospatial intelligence, bioinformatics (e.g. genomics), finance, scientific computing, social network analysis, recommendation systems, and many others. Kolda et al. have published a survey on various tensor decompositions and their applications [1].

Data analysis through tensor decompositions involves intensive computations over large-scale irregular sparse data. Optimizing the execution of such data intensive computations is key to reducing the time-to-solution (or response time) in real-world data analysis applications. Additionally, the volume of data processed in critical applications is growing in size. This growth is adding complexity with respect to quicker turnaround and feasibility to handle large data, in data analysis.

As HPC systems are increasingly used for data analysis applications, it is becoming increasingly important to optimize sparse tensor computations and execute them efficiently on modern and advanced HPC systems such as multi-core, many-core, and future deep memory hierarchy exascale systems. In addition to utilizing the large processing capability of HPC systems, it is crucial to improve memory performance (memory usage, communication, synchronization, memory reuse, and data locality) in HPC systems.

In this paper, we present multiple optimizations for sparse tensor computations that are targeted towards faster and memory-efficient execution on HPC systems. We present a technique to significantly reduce memory usage in tensor decompositions through rematerialization of computations. This optimization is key to applying tensor decompositions for large problems that would be otherwise infeasible on an HPC multi-core system. We also present an approach for performing operation- and memory-efficient sparse tensor computations that broadly applies to three different CP tensor decomposition methods. Our approach exploits the redundancy in the non-zero structure of sparse tensors and reuses partial results in tensor computations with insignificant memory overhead. Further, we introduce an optimization that fuses sparse tensor and matrix operations and enables increased thread-local computations, reduced synchronizations (and communications), and improved data locality and reuse.

Overall, we make the following contributions in this paper:

1) Algorithmic improvements to three different widely applicable CP tensor decomposition methods (and not just one particular method that is not necessarily useful for all applications) to increase the feasibility and breadth of application of tensor analysis
2) Improvements to reduce memory usage, synchronizations, and execution time of tensor decomposition methods, and enable the use of powerful tensor decomposition methods in critical real-world applications across multiple domains
3) Demonstrate how our improvements benefit tensor analysis of datasets from cybersecurity, geospatial intelligence, bioinformatics, Natural Language Processing (NLP), and social network domains.

We integrate the optimizations presented in this paper into ENSIGN [2], a commercially available tensor decomposition package containing high-performance C implementations of a variety of tensor decomposition methods. ENSIGN tensor methods are implemented using optimized sparse tensor data structures, namely, mode-specific sparse (MSS) tensor and

DOI: 10.1109/HPEC.2017.8091026

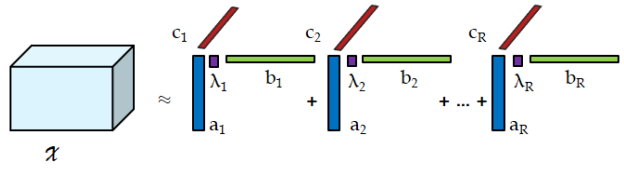


Fig. 1. CP decomposition

mode-generic sparse (MGS) tensor data structures [3], and are parallelized and optimized for load balancing and data locality [3], [4], [5], [6]. The data structures used in ENSIGN provide a foundation for a broad range of decomposition methods and tensor sizes, and are not specifically optimized for any particular method. In this work, we make use of the MSS and MGS data structures in our optimized tensor decompositions, and build on top of the optimizations that are already implemented with the tensor methods.

II. Background

In this section, we give some background information that will help to understand our techniques and experimental study in this work. We discuss the basic definitions and algorithms for tensor decompositions.

A tensor is a multi-dimensional array and the *order* of a tensor is the number of dimensions, also called *modes*, of the tensor. A tensor can be transformed into a matrix (i.e. *matricized*) by flattening it along any of its modes. The mode-n matricized form of $\mathcal{X}$ is denoted by $\mathbf{X}_{(n)}$.

There are two popular and prominent tensor decomposition models, namely, CANDECOMP/PARAFAC (CP) and Tucker decompositions. We will focus our discussion in this paper to CP decomposition, especially, three CP decomposition algorithms that are useful in real applications.

The CP decomposition decomposes a tensor into a sum of component rank-one tensors (a N-way tensor is called a rank-one tensor if it can be expressed as an outer product of N vectors). This is illustrated in Figure 1. The CP decomposition that factorizes an input tensor $\mathcal{X}$ of size $I_1 \times \cdots \times I_N$ into R components (with factor matrices $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$ and weight vector λ) is of the form: $\mathcal{X} = \sum_{r=1}^{R} \lambda_r \mathbf{a}_r^{(1)} \circ \cdots \circ \mathbf{a}_r^{(N)}$ where $\mathbf{a}_r^{(n)}$ represents the r^{th} column of the factor matrix $\mathbf{A}^{(n)}$ of size $I_n \times R$.

a) CP-ALS Algorithm: The widely used workhorse algorithm for CP decomposition is the alternating least squares (ALS) method (presented in Algorithm 1).

b) CP-ALS-NN Algorithm: The algorithm for non-negative (NN) CP-ALS decomposition using multiplicative updates [7] is similar to Algorithm 1, but differs only in the way the factor matrices are updated at each step (Line 6). Line 6 of CP-ALS-NN algorithm looks like:

$$\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \frac{\mathbf{U}}{\mathbf{A}^{(n)}\mathbf{V}}$$

c) CP-APR Algorithm: Chi and Kolda [8] have developed an Alternate Poisson Regression (APR) fitting algorithm for non-negative CP tensor decomposition that is well-suited for modeling and analyzing sparse count data. The CP-APR algorithm described in [8] is presented in Algorithm 2.

Algorithm 1 CP-ALS Algorithm

1: initialize $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
2: **repeat**
3: **for** $n = 1 \ldots N$ **do**
4: $\mathbf{V} = *_{m \neq n} \mathbf{A}^{(m)T} \mathbf{A}^{(m)}$
5: $\mathbf{U} = \mathbf{X}_{(n)} (\odot_{m \neq n} \mathbf{A}^{(m)})$
6: $\mathbf{A}^{(n)} = \mathbf{U}\mathbf{V}^{\dagger}$
7: **end for**
8: **until** convergence

Algorithm 2 CP-APR Algorithm

1: initialize $\mathbf{A}^{(1)} \ldots \mathbf{A}^{(N)}$
2: **repeat**
3: **for** $n = 1 \ldots N$ **do**
4: $\mathbf{\Pi} = (\odot_{m \neq n} \mathbf{A}^{(m)})^T$
5: **repeat**
6: $\mathbf{\Phi} = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)} \mathbf{\Pi})) \mathbf{\Pi}^T$
7: $\mathbf{A}^{(n)} = \mathbf{A}^{(n)} * \mathbf{\Phi}$
8: **until** convergence
9: **end for**
10: **until** convergence

The sparse computations that we focus and that are computationally expensive in these algorithms are:

- Sparse matricized tensor times Khatri-Rao product (MTTKRP) computation in CP-ALS and CP-ALS-NN algorithms: $\mathbf{U} = \mathbf{X}_{(n)} (\odot_{m \neq n} \mathbf{A}^{(m)})$
- Sparse Khatri-Rao product ($\mathbf{\Pi}$) computation in CP-APR algorithm: $\mathbf{\Pi} = (\odot_{m \neq n} \mathbf{A}^{(m)})^T$
- Sparse $\mathbf{\Phi}$ computation (a composite operation involving elementwise division, matrix multiplication, tensor vector product) in CP-APR algorithm: $\mathbf{\Phi} = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)} \mathbf{\Pi})) \mathbf{\Pi}^T$

In the sparse version of a Khatri-Rao product (KRP) of M matrices ($\mathbf{A}^{(1)}, \ldots, \mathbf{A}^{(M)}$, the size of the nth matrix being $I_n \times R$, $1 \leq n \leq M$), we need not compute all the $\Pi_{n=1}^{N} I_n$ rows, but only certain number of rows driven by the non-zero structure (number of non-zeros, say P, and list of non-zero indices) of a sparse tensor along which the KRP is used in other computations in the decomposition method. Hence, sparse KRP computation involves only a selected number of rows, usually, one per non-zero of the sparse tensor involved in the decomposition. Each row of the sparse KRP is given by the elementwise product (also called the Hadamard product) of the rows of the matrix selected by the index of the non-zero. If the p-th non-zero index of the tensor is $\langle i_1, \ldots, i_M \rangle$, then the p-th row of sparse KRP is given by:

$$\mathbf{\Pi}(p,:) = \mathbf{A}^{(1)}(i_1,:) * \cdots * \mathbf{A}^{(M)}(i_M,:)$$

III. Related Work

There are several works that have focused on optimizing sparse MTTKRP for shared- and distributed-memory systems. However, we are not aware of any work on optimizing CP-APR, other than our previous work [4]. In our previous work, we presented a NUMA-aware low-overhead scheduler for load balancing sparse tensor operations. The work presented in this work is complementary to our previous work. Kolda et al. develop and deliver a MATLAB Tensor Toolbox [9] that provides a broad range of tensor decomposition methods, but they cannot be used for HPC purposes. Plantenga and Kolda provide a C++ based Tensor Toolbox [10], but the tensor decomposition methods are not optimized for efficient parallel execution. Ravindran et al. [11] presented an MTTKRP formulation for three-mode tensors that achieves computational savings but restricted to three modes. Smith et al. [12], [13] provide a tool called SPLATT in which they have a high-performance MPI+OpenMP implementation of CP-ALS by optimizing sparse MTTKRP and other computations in CP-ALS. SPLATT does not support CP-APR or CP-ALS-NN. SPLATT uses a compressed sparse tensor data structure called "compressed sparse fiber" (CSF) [14] to identify and eliminate redundancies (using the sparsity pattern of tensors) in MTTKRP computation. Choi et al. [15] (through their tool DFacTo) and Kaya et al. [16] (through their tool HyperTensor) provide a distributed-memory MPI version of CP-ALS, but they also do not support CP-APR or CP-ALS-NN. Rolinger et al. [17] performed an empirical evaluation of the performance of CP-ALS implementation of the different tools, namely, ENSIGN, SPLATT, and DFacTo, and presented the strengths and areas of improvement of each tool.

IV. Techniques for Improving Sparse Tensor Computations

We present a detailed description of our approach in which we develop multiple optimizations aimed at improving the performance in terms of execution time and memory usage of tensor decompositions.

For further discussion, assume that a sparse tensor that is decomposed by the existing and optimized tensor decomposition methods (that we focus in this paper) has N modes and P non-zero values. Let the size of the tensor be $I_1 \times \cdots \times I_N$. Let the tensor be decomposed into R components.

A. Selective rematerialization of sparse Khatri-Rao product

A critical challenge in scaling and applying the CP-APR method for tensor analysis is addressing the memory blowup problem. The matrix $\mathbf{\Pi}$, storing the result of sparse KRP, that is computed in Line 4 of Algorithm 2 is of size $P \times R$. Since $\mathbf{\Pi}$ is reused in the "inner iterative loop" starting in Line 5 of Algorithm 2, the existing state-of-the-art implementation of CP-APR computes $\mathbf{\Pi}$ as a full $P \times R$ matrix. If the tensor is large (which typically means that P is large) or if the tensor is decomposed into very high number of components (in other words, if R is large), this **adds a huge memory pressure** or **makes it infeasible to perform the tensor decomposition**.

Our solution to address this problem is to fold in the computation of $\mathbf{\Pi}$ into the inner iterative loop and recompute the rows of $\mathbf{\Pi}$ as they are needed in the computation of $\mathbf{\Phi}$ (Line 6 of Algorithm 2). We call this "selective rematerialization" of sparse KRP. As a result, in Algorithm 2, Line 4 is (removed and) folded into Line 6, and Line 6 becomes:

$$\mathbf{\Phi} = (\mathbf{X}_{(n)} \oslash (\mathbf{A}^{(n)}(\odot_{m \neq n} \mathbf{A}^{(m)})))(\odot_{m \neq n} \mathbf{A}^{(m)})^T \quad (1)$$

Selective rematerialization (or recomputation) of rows of sparse KRP decreases memory pressure and memory usage from $\mathcal{O}(PR)$ to $\mathcal{O}(P)$. However, it increases the amount of computations that are performed in $\mathbf{\Phi}$ computation. Computation of $\mathbf{\Phi}$ is the most computationally expensive step ($\mathcal{O}(PR)$ for each inner iteration) in CP-APR and it becomes even more expensive ($\mathcal{O}(PNR)$ for each inner iteration) with rematerialization of $\mathbf{\Pi}$. Section IV-B describes an optimization in which we exploit the redundancy in the non-zero structure (i.e. redundancy of tensor indices) of the sparse tensor and optimize the unified computation of $\mathbf{\Pi}$ and $\mathbf{\Phi}$ described in Equation 1.

B. Exploiting the redundancy in non-zero structure of tensors

Sparse tensors are usually represented as a list of values and multi-indices (called the "coordinate format" of sparse tensors). There is a lot of redundancy in the tensor indices since tensor entries have common sub-indices. For example, if $(\langle 1,1,1 \rangle, 5.0)$ and $(\langle 1,1,2 \rangle, 5.2)$ are two tensor entries in a sparse tensor, they have common sub-indices $\langle 1,1 \rangle$ in the first two modes. ENSIGN MSS and MGS sparse tensor formats exploit this redundancy and provide a compressed tensor storage. We now discuss how we exploit the redundancy in non-zero structure (along with ENSIGN MGS and MSS data structures) and eliminate redundant computations through reuse of partial results from common sub-expressions (with insignificant memory overhead) in tensor decompositions, especially, computations involving sparse KRP.

Recalling the discussion from Section II, we compute P rows of sparse KRP corresponding to P non-zeros of the sparse tensor and row p ($1 \leq p \leq P$) of sparse KRP is given by the elementwise (Hadamard) product of the rows of the matrix selected by the index of the p-th non-zero. We now discuss how we perform the "computation-minimal" unified and rematerialized $\mathbf{\Pi}$ and $\mathbf{\Phi}$ computation in the modified CP-APR algorithm.

Figure 2 illustrates our approach to eliminate redundant computations in sparse KRP operation. We use N KRP buffers each of size R, one corresponding to each mode. For large tensors, the amount of computation savings achieved from this optimization outweighs the $\mathcal{O}(NR)$ memory overhead introduced for the buffers. With the ENSIGN MSS and MGS tensor data structures, the indices are ordered such that consecutive non-zeros have common sub-indices as much as possible (with the indices diverging from the innermost mode to the outermost mode along the list of non-zeros) as illustrated in Figure 2. This enables reuse of partial results in the computation of sparse KRP.

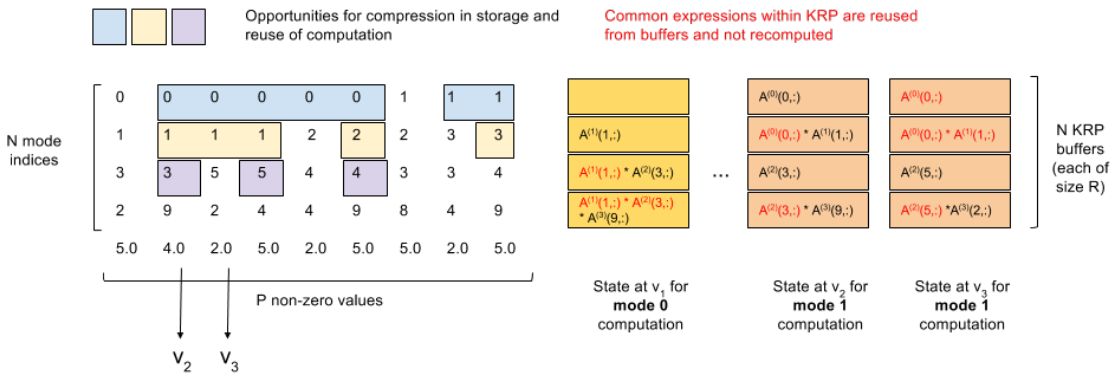


Fig. 2. Illustrative example describing our approach for efficient sparse KRP: Computations that are directly or indirectly driven by the tensor benefit from KRP buffers that store partial KRP results. Tensor elements that share common sub-indices enable sharing and reuse of buffered partial KRP results corresponding to the common sub-indices. This reduces computations and avoids accessing factor matrices that are irregularly accessed.

Algorithm 3 presents the computation-minimal unified computation of $\mathbf{\Phi}$ and $\mathbf{\Pi}$ (i.e. optimized version of Line 6 in Algorithm 2 after it is modified as per Equation 1). It assumes the following: $\mathbf{S}$ stores the indices of the sparse tensor, $\mathbf{v}$ stores the values of the tensor, and $\mathbf{K}$ is the $N \times R$ KRP buffer matrix. It reduces the complexity of computation from $\mathcal{O}(PNR)$ for each inner iteration to $\mathcal{O}(PN_rR)$ for each inner iteration, where $N_r = \psi_r N, 0 < \psi_r \leq 1$ and ψ_r represents the redundancy factor inherent in the structure of the non-zero.

An important observation is that changing Line 7 to $w = 1$ in Algorithm 3 makes it computation-minimal sparse MTTKRP (i.e. optimized version of Line 5 in Algorithm 1) for CP-ALS and CP-ALS-NN.

Algorithm 3 Unified and optimized $\mathbf{\Phi}$ and $\mathbf{\Pi}$ computation for mode n in CP-APR

1: $\mathbf{b} = \mathbf{0}$
2: **for** $p = 1 \ldots P$ **do**
3: $q = \begin{cases} \text{diverging mode}(\mathbf{S}(p,:), \mathbf{S}(p-1,:)), \text{if } p > 1 \\ 1, \text{if } p = 1 \end{cases}$
4: **for** $m = q \ldots N$ **do**
5: $\mathbf{K}(m,:) = \begin{cases} \mathbf{A}^{(m)}(\mathbf{S}(p,m),:), \text{if } m = 1 \text{ or} \\ \quad (m = n+1 \text{ and } m \neq N) \\ \mathbf{K}(m-1,:) * \mathbf{A}^{(m)}(\mathbf{S}(p,m),:), \text{else} \end{cases}$
6: **end for**
7: $w = \begin{cases} \mathbf{K}(n,:).\mathbf{K}(N,:), \text{if } n \neq N \\ \mathbf{K}(n,:).\mathbf{1}_R, \text{if } n = N \end{cases}$
8: $u = \mathbf{v}(p)/w$
9: **if** $q \leq n$ **then**
10: $\mathbf{\Phi}(n,:) += \mathbf{b}$
11: $\mathbf{b} = \mathbf{0}$
12: **end if**
13: $\mathbf{b} += \begin{cases} u * \mathbf{K}(N,:), \text{if } n = 1 \\ u * \mathbf{K}(n-1,:), \text{if } n = N \\ u * \mathbf{K}(N,:) * \mathbf{K}(n-1,:), \text{else} \end{cases}$
14: **end for**

C. Fusion of sparse tensor and matrix operations

It is extremely important to exploit the large processing capability (improve concurrency and load balance) and memory hierarchy (increase accesses to faster memory and reduce data movement) in modern and advanced HPC systems such as multi-core, many-core, and deep hierarchy exascale systems. Towards this objective, we present a technique that (1) increases thread-local computations and exploits parallelism, (2) reduces synchronizations and communications, and (3) improves data locality and reuse.

Our approach is to efficiently and carefully fuse the sparse tensor and matrix operations such that a processing thread performs more local computations and less synchronizations (global or point-wise) and brings the consumption of results of fused composite operations closer to their production.

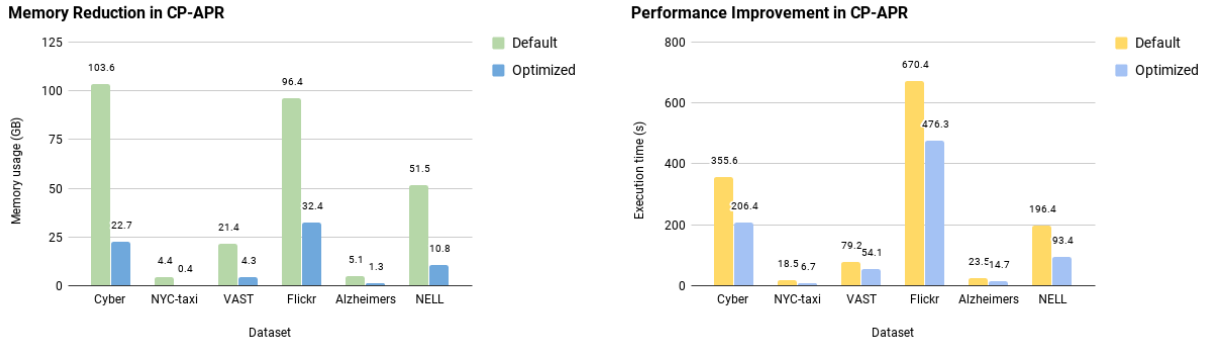
In the CP-APR algorithm, we fold Line 4 into Line 6 (because of rematerialization described in Section IV-A) and fuse Lines 6 (that is optimized as described in Section IV-B) and 7 into one composite operation. Similarly, for CP-ALS (and CP-ALS-NN), we fuse Lines 5 (that is optimized as described in Section IV-B) and 6 in Algorithm 1 into one composite operation.

Without the fusion of operations described in our approach, each operation will be "individually" optimized and parallelized (for example using OpenMP parallel construct), with synchronizations between the individual parallelized operations. If the pattern of distribution of computations across processor cores is different across the operations, then it would also make the processing thread access non-local data resulting from the previous operation.

V. Experimental Results

We provide a detailed experimental study on the evaluation of our techniques to improve the memory efficiency and parallel performance and scalability of sparse tensor computations on multi-core systems.

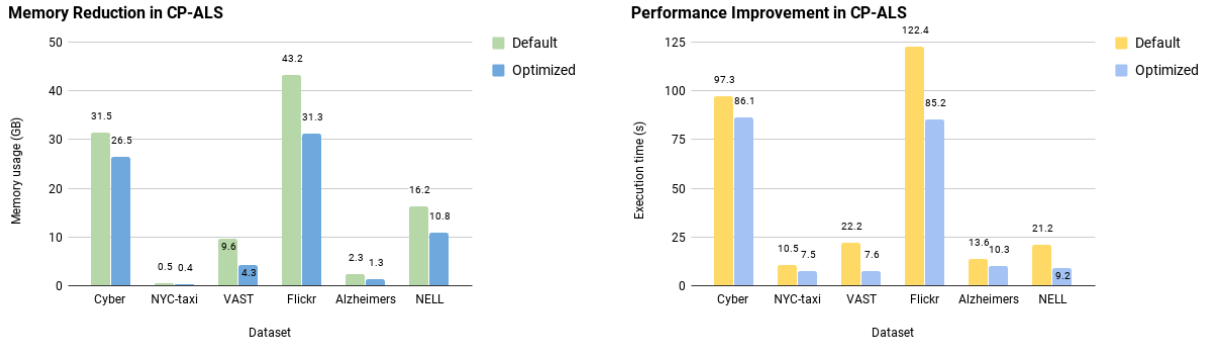
Datasets: We use multiple real datasets from different application domains to evaluate the techniques described in our work. Table I describes the datasets. These datasets are of



(a) Memory reduction

(b) Performance improvement

Fig. 3. Performance efficiency of CP-APR on different datasets on 32 cores



(a) Memory reduction

(b) Performance improvement

Fig. 4. Performance efficiency of CP-ALS on different datasets on 32 cores

Dataset	NNZ	Tensor Dimensions	Domain
Cyber [18]	87M	[10M, 9K, 71K, 40K]	Cybersecurity
NYC-taxi [19]	2M	[168, 15K, 38K]	Geospatial
VAST [20]	26M	[165K, 11K, 2, 100, 89]	Geospatial
Flickr [21]	112M	[319K, 28M, 1.6M, 731]	Social network
Alzheimers [22]	6M	[5, 1309, 156, 1047, 396]	Bioinformatics
NELL [23]	76M	[12K, 9K, 28K]	NLP

TABLE I
DATASETS USED IN OUR EVALUATION

different size and non-zero structure. These datasets represent a good mix of datasets that are being used by Reservoir Labs for real critical analysis in the fields of cybersecurity and bioinformatics (e.g. Cyber and Alzheimers datasets) and datasets that are being used by the tensor community [24]. Different CP decomposition methods are suited for different types of datasets and/or different domains and give qualitatively different results. In particular, we have seen CP-APR to be very useful for analyzing sparse count data and hence suited for cybersecurity and geospatial intelligence analysis. For our experiments, we use a decomposition rank of 30 for all datasets except NYC-taxi (rank of 100) and Cyber (rank of 50) datasets.

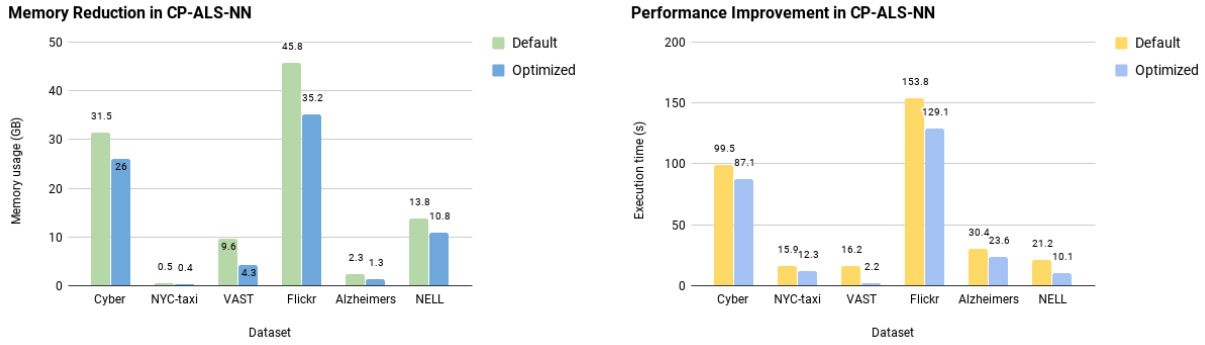
Experimental system: We use a modern multi-core system to evaluate our techniques. The system we use is a quad socket 8-core system with Intel Xeon E5-4620 2.2 GHz processors (Intel Sandy Bridge microarchitecture chips). The system has 128 GB of DRAM.

Baseline and optimized versions for evaluation: The baseline versions of CP-APR, CP-ALS, and CP-ALS-NN methods are the latest "default" versions of these methods in ENSIGN without the optimizations discussed in the paper. The default versions in ENSIGN are highly optimized for parallelism and locality as mentioned in Section I. The "optimized" versions include the optimizations discussed in the paper such as selective rematerialization of sparse KRP (for CP-APR), exploitation of redundancy in non-zero structure, and fusion of sparse tensor and matrix operations.

A. *Memory efficiency*

We characterize memory usage in the core computational portion (i.e. the significant portion of execution) of the three tensor decomposition methods. We measure maximum "resident set size" (RSS) to characterize memory usage. Figures 3(a), 4(a), and 5(a) show the memory usage by CP-APR, CP-ALS, and CP-ALS-NN, respectively, on different datasets when these methods are executed on 32 cores. The figures clearly illustrate significant gains in terms of reduction in memory usage achieved using our techniques.

The most beneficial method in terms of memory efficiency is CP-APR due to selective rematerialization of sparse KRP. The NYC-taxi dataset has a significant 11x reduction in memory usage, while other datasets have a 3-5x reduction. More importantly, without the memory-efficient optimization, we would not have been able to apply CP-APR for datasets



(a) Memory reduction

(b) Performance improvement

Fig. 5. Performance efficiency of CP-ALS-NN on different datasets on 32 cores

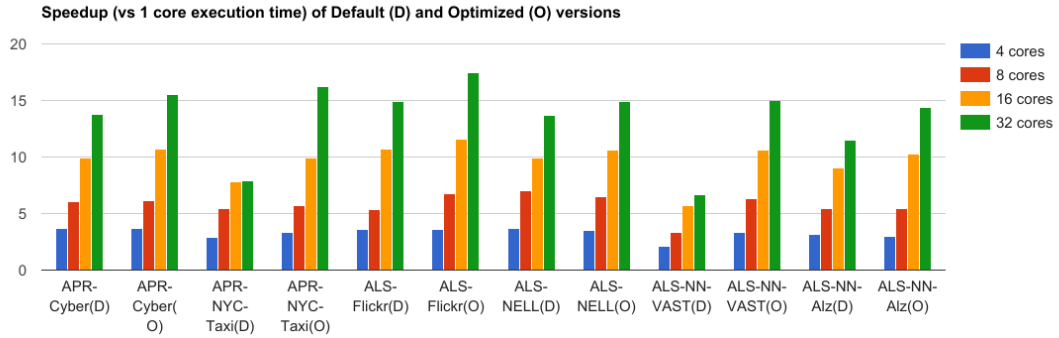


Fig. 6. Speedup of CP-APR, CP-ALS, and CP-ALS-NN optimized versions (and default versions — for comparison) on different datasets across 32 cores

such as Cyber and Flickr for a larger decomposition rank (rank ≥ 85 for Cyber and rank ≥ 65 for Flickr).

The memory usage characteristics of CP-ALS and CP-ALS-NN are very similar, as they are similar algorithmically. The memory reduction is between 20% and 70% for all datasets, except VAST that has a 2x reduction.

B. Performance improvement

We characterize the performance of the three tensor decomposition methods by measuring the overall execution time of the methods (and not just the core computation kernels). Figures 3(b), 4(b), and 5(b) show the performance improvement in terms of reduced execution time achieved using our techniques.

The NELL dataset has significant improvement (2x) across all methods because the redundancy in non-zero structure of the dataset is better exploited. For CP-APR, the NYC-taxi geospatial dataset achieves 2.75x and the Cyber dataset achieves 1.75x improved performance. The other datasets achieve a 40-50% improvement. For CP-ALS and CP-ALS-NN, the VAST dataset achieves significant improvement (3x for ALS and 7x for ALS-NN). The other datasets achieve a 20-40% improvement.

Overall, the reduced memory pressure reduces cache misses and processor stall cycles and leads to improved execution time. Further, efficient sparse KRP computation also reduces the number of computations (significantly, in some datasets) and associated memory accesses and contributes to improved execution. Also, the parallel performance and speedup are improved due to reduced synchronizations and improved data locality resulting from fusion of tensor-matrix operations.

C. Scalability

The optimized version achieves better speedup than the default version in all methods across all datasets. Figure 6 shows the speedup (on a selected datasets-to-methods combination) achieved by the default and optimized versions on 4, 8, 16, and 32 cores compared to 1 core. The selection is based on the "practical" fit of the tensor decomposition method on the dataset or application domain. We observe around 15-16x speedup factor on 32 cores. The speedup improvement of optimized versions compared to default versions is around 10-25% for all datasets except NYC-taxi and Flickr datasets that show 2x improvement.

VI. Conclusion

In this paper, we have developed techniques that will enable the use of powerful tensor decomposition methods in critical real-world applications across multiple domains such as cyber-security, geospatial intelligence, bioinformatics, NLP, and social network analysis. We have implemented high-performance tensor decomposition methods that will scale well on modern and advanced HPC systems. We are currently developing communication-minimal distributed-memory implementations of tensor decomposition methods using the memory-efficient, synchronization-minimal, and parallelism-efficient techniques presented in this paper.

REFERENCES

[1] T. G. Kolda and B. W. Bader, "Tensor Decompositions and Applications," *SIAM Review*, vol. 51, no. 3, pp. 455–500, September 2009.

[2] R. Labs, "ENSIGN Tensor Toolbox." [Online]. Available: https://www.reservoir.com/support/ensign/

[3] M. Baskaran, B. Meister, N. Vasilache, and R. Lethin, "Efficient and Scalable Computations with Sparse Tensors," in *IEEE High Performance Extreme Computing Conference*, Waltham, MA, Sep. 2012.

[4] M. Baskaran, B. Meister, and R. Lethin, "Low-overhead Load-balanced Scheduling for Sparse Tensor Computations," in *IEEE High Performance Extreme Computing Conference*, Waltham, MA, Sep. 2014.

[5] M. M. Baskaran, B. Meister, and R. Lethin, "Parallelizing and Optimizing Sparse Tensor Computations," in *Proceedings of the 28th ACM International Conference on Supercomputing*, ser. ICS '14, 2014, pp. 179–179.

[6] J. Cai, M. Baskaran, B. Meister, and R. Lethin, "Optimization of Symmetric Tensor Computations," in *IEEE High Performance Extreme Computing Conference*, Waltham, MA, Sep. 2015.

[7] D. Chen and R. J. Plemmons, "Nonnegativity constraints in numerical analysis," in *Symposium on the Birth of Numerical Analysis*, 2007.

[8] E. C. Chi and T. G. Kolda, "On Tensors, Sparsity, and Nonnegative Factorizations," arXiv:1304.4964 [math.NA], December 2011. [Online]. Available: http://arxiv.org/abs/1112.2414

[9] T. G. Kolda and B. Bader, "MATLAB Tensor Toolbox." [Online]. Available: http://www.sandia.gov/tgkolda/TensorToolbox

[10] T. D. Plantenga and T. G. Kolda, "C++ Tensor Toolbox." [Online]. Available: http://prod.sandia.gov/techlib/access-control.cgi/2012/123087.pdf

[11] N. Ravindran, N. D. Sidiropoulos, S. Smith, and G. Karypis, "Memory-efficient parallel computation of tensor and matrix products for big tensor decomposition," in *Asilomar Conference on Signals, Systems, and Computers*, 2014.

[12] S. Smith and G. Karypis, "A Medium-Grained Algorithm for Distributed Sparse Tensor Factorization," in *Parallel and Distributed Processing Symposium (IPDPS), 2016 IEEE International*. IEEE, 2016.

[13] S. Smith, N. Ravindran, N. D. Sidiropoulos, and G. Karypis, "SPLATT: Efficient and parallel sparse tensor-matrix multiplication," in *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*. IEEE, 2015, pp. 61–70.

[14] S. Smith and G. Karypis, "Tensor-matrix products with a compressed sparse tensor," in *Proceedings of the 5th Workshop on Irregular Applications: Architectures and Algorithms*. ACM, 2015, p. 7.

[15] J. H. Choi and S. Vishwanathan, "DFacTo: Distributed factorization of tensors," in *Advances in Neural Information Processing Systems 27*, Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2014, pp. 1296–1304. [Online]. Available: http://papers.nips.cc/paper/5395-dfacto-distributed-factorization-of-tensors.pdf

[16] O. Kaya and B. Uçar, "Scalable sparse tensor decompositions in distributed memory systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '15. New York, NY, USA: ACM, 2015, pp. 77:1–77:11. [Online]. Available: http://doi.acm.org/10.1145/2807591.2807624

[17] T. B. Rolinger, T. A. Simon, and C. D. Krieger, "Performance Evaluation of Parallel Sparse Tensor Decomposition Implementations," in *Proceedings of the Sixth Workshop on Irregular Applications: Architectures and Algorithms*, 2016, pp. 54–57.

[18] R. Labs, "R-Scope Network Sensor." [Online]. Available: https://www.reservoir.com/product/r-scope/

[19] N. Taxi and L. Commission, "New york city taxi and limousine commision (tlc) trip record data." [Online]. Available: http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml

[20] V. A. Community, "Vast challenge 2015 mini-challenge 1." [Online]. Available: http://vacommunity.org/VAST+Challenge+2015

[21] O. Görlitz, S. Sizov, and S. Staab, "Pints: peer-to-peer infrastructure for tagging systems." in *IPTPS*, 2008, p. 19.

[22] B. Institute, "Connectivity map." [Online]. Available: https://portals.broadinstitute.org/cmap/

[23] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. R. Hruschka, Jr., and T. M. Mitchell, "Toward an architecture for never-ending language learning," in *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, ser. AAAI'10. AAAI Press, 2010, pp. 1306–1313. [Online]. Available: http://dl.acm.org/citation.cfm?id=2898607.2898816

[24] S. Smith, J. W. Choi, J. Li, R. Vuduc, J. Park, X. Liu, and G. Karypis. (2017) FROSTT: The formidable repository of open sparse tensors and tools. [Online]. Available: http://frostt.io/