**Parallel Scientific Computing**
**May 30, 2021**

# ID5130

# PARALLEL ANALYSES OF CANONICAL POLYADIC TENSOR DECOMPOSITION ALGORITHM

**G Prashant (BS17B011)**
Department of Biotechnology
Indian Institute of Technology Madras
Email: bs17b011@smail.iitm.ac.in

**Burhanuddin Sabuwala (BE17B011)**
Department of Biotechnology
Indian Institute of Technology Madras
Email: burhan.sabuwala@smail.iitm.ac.in

## ABSTRACT

A tensor is a higher-order array of multiple dimensions. Tensors have unique properties, most of which are not coherent with those of matrices. Hence, specialized algorithms are required for handling and processing multidimensional tensors. In this regard, tensor decomposition plays an instrumental role in identifying meaningful and interpretable insights from data. One of the most prominent tensor decomposition algorithms is CP (Canonical Polyadic or CANDECOMP/PARAFAC) decomposition, which decomposes a tensor into a sum of rank-1 tensors. While there are multiple versions of the CP decomposition algorithm, we look forward to parallelizing the most straightforward approach that uses alternating least squares (CP-ALS) for optimization [1]. We have leveraged OpenMP and OpenACC for parallelizing the algorithm. We tested our serial and parallel code on synthetically generated data.

## NOMENCLATURE

$\mathcal{T}$ A tensor is represented as an italicized character
$\boldsymbol{A}$, $L$ Matrices are represented either in bold or non-bold, this way
$a$ A vector is represented in smaller case
$\mathcal{T}_{i,j,k}$ Tensor is indexed this way
$\boldsymbol{A}_{i,j}$ A matrix is indexed this way
$\boldsymbol{A} \circ \boldsymbol{B}$ Hadamard Product (Element-wise multiplication)
$a \otimes b$ Outer product of two vectors
$a \otimes b \otimes c \cdots$ Outer product of more than two vectors
$\boldsymbol{A} \odot \boldsymbol{B}$ The Khatri-Rao product
$\mathcal{T}^{(k)}$ $k^{th}$ unfolding of a tensor - matricising a tensor along the $k^{th}$ dimension

## INTRODUCTION

Tensors are higher-dimensional arrays, with vectors and matrices being the simplest tensors with one and two dimensions, respectively. A tensor with $N$ dimensions can be denoted as $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times \cdots \times d_N}$, where $a_1, \cdots, a_N$ are the sizes of each dimension. Each element in a tensor of order $N$ can be accessed by a set of $N$ integers $i_1, \cdots, i_N$ such that $1 \leq i_k \leq d_k \; \forall \; k \in \{1, \cdots, N\}$. Data in the form of higher dimensional tensors are extremely prominent in bioinformatics, signal processing, image analysis, network science, and deep learning. It is important to note that tensors have distinct properties that are not coherent with those of matrices. To derive meaningful information from large tensors and facilitate machine learning analysis, it is necessary to decompose or factorize the tensors into simpler tensors. Decomposing tensors have successfully provided a way to extract domain-specific information from data in the areas of neuroscience [2] and social network analysis [3].

### Matrix Decomposition

For a given matrix $\boldsymbol{M}$, it is possible to decompose (or factorize) it into two component matrices $\boldsymbol{A}$, $\boldsymbol{B}$ such that

$$\boldsymbol{M} = \boldsymbol{A}\boldsymbol{B}^T \tag{1}$$

Here, $\boldsymbol{A}$ and $\boldsymbol{B}$ need not be unique factor matrices. To illustrate the same, consider a orthogonal rotation matrix $\boldsymbol{R}$, in which case $\boldsymbol{R}\boldsymbol{R}^{\boldsymbol{T}} = \boldsymbol{I}$. By incorporating $\boldsymbol{R}$ in equation (1), we get

$$\boldsymbol{M} = (\boldsymbol{A}\boldsymbol{R})(\boldsymbol{R}^{\boldsymbol{T}}\boldsymbol{B}^{\boldsymbol{T}}) = (\boldsymbol{A}\boldsymbol{R})(\boldsymbol{B}\boldsymbol{R})^T \tag{2}$$

implying that there are multiple ways to decompose or factorize a matrix. The decomposition can however be unique if stringent conditions are imposed on the factor matrices $\boldsymbol{A}$ and $\boldsymbol{B}$. Having that said, algorithms like LU decomposition, QR decomposition, Singular Value Decomposition (SVD) are systematic and meaningful ways to decompose a matrix uniquely by enforcing appropriate conditions. For instance, in QR decomposition, $\boldsymbol{A}$ is enforced to be an orthogonal matrix and $\boldsymbol{B}$ is enforced to be a lower triangular matrix.

### Tensor Decomposition

Tensor decomposition, or precisely, tensor rank decomposition is the technique of factorizing a tensor into a sum of low-rank tensors. One of the most prominent tensor decomposition algorithms is the CP (Canonical Polyadic or CANDECOMP/PARAFAC) decomposition algorithm. The goal of the CP algorithm is to construct vectors $a_i^{(r)}, \forall\ i \in \{1,\cdots,N\}$ and $\forall\ r \in \{1,\cdots,R\}$ such that

$$\mathcal{T} \approx \hat{\mathcal{T}} = \sum_{r=1}^{R} a_1^{(r)} \otimes a_2^{(r)} \otimes \cdots \otimes a_N^{(r)} \tag{3}$$

where $R$ is the rank of the tensor. Here, $a_i^r \in \mathbb{R}^{d_i}\ \forall\ r$. The goal of most of the implementations of the CP algorithm is to minimize the least squares objective, as given in equation (4).

$$\mathcal{J} = \min_{a_i^{(r)}} \|\mathcal{T} - \hat{\mathcal{T}}\|_F^2 \tag{4}$$

### Alternating Least Squares (CP-ALS)

For simplicity, consider a third order tensor $\mathcal{T} \in \mathbb{R}^{d_1 \times d_2 \times d_3}$ (3-dimensional array), where we are required to find vectors $a_i^{(r)}$ such that,

$$\mathcal{T} \approx \hat{\mathcal{T}} = \sum_{r=1}^{R} a_1^{(r)} \otimes a_2^{(r)} \otimes a_3^{(r)} \tag{5}$$

Now, let us represent factor matrices $\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$ as follows,

$$\boldsymbol{A} = \begin{bmatrix} a_1^{(1)} & \cdots & a_1^{(R)} \end{bmatrix}_{d_1 \times R} \tag{6}$$

$$\boldsymbol{B} = \begin{bmatrix} a_2^{(1)} & \cdots & a_2^{(R)} \end{bmatrix}_{d_2 \times R} \tag{7}$$

$$\boldsymbol{C} = \begin{bmatrix} a_3^{(1)} & \cdots & a_3^{(R)} \end{bmatrix}_{d_3 \times R} \tag{8}$$

The optimization objective hence becomes

$$\mathcal{J} = \min_{\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}} \sum_{i,j,k} \left(\mathcal{T}_{i,j,k} - \sum_{r=1}^{R} \boldsymbol{A}_{ir}\boldsymbol{B}_{jr}\boldsymbol{C}_{kr}\right)^2 \tag{9}$$

The idea behind the alternating least-squares (ALS) approach is cast the complex nonconvex optimization problem given in equation (9) into three simpler and convex least squares optimization problems as follows,

Step 1: Keeping $\boldsymbol{B}, \boldsymbol{C}$ fixed, we can solve for $\boldsymbol{A}$ as

$$\begin{aligned} \mathcal{J}_A &= \min_{\boldsymbol{A}} \sum_{i,j,k} \left(\mathcal{T}_{i,j,k} - \sum_{r=1}^{R} \boldsymbol{A}_{ir}\boldsymbol{B}_{jr}\boldsymbol{C}_{kr}\right)^2 \\ &= \min_{\boldsymbol{A}} \|\mathcal{T}^{(1)} - \boldsymbol{A}(\boldsymbol{C} \odot \boldsymbol{B})\|_F^2 \end{aligned} \tag{10}$$

Solving for $\boldsymbol{A}$, we get,

$$\boldsymbol{A} = \left(\mathcal{T}^{(1)}(\boldsymbol{C} \odot \boldsymbol{B})\right)(\boldsymbol{C}^T\boldsymbol{C} \circ \boldsymbol{B}^T\boldsymbol{B})^{-1} \tag{11}$$

Step 2: Similarly, keeping $\boldsymbol{C}, \boldsymbol{A}$ fixed, we can solve for $\boldsymbol{B}$ as

$$\begin{aligned} \mathcal{J}_B &= \min_{\boldsymbol{B}} \sum_{i,j,k} \left(\mathcal{T}_{i,j,k} - \sum_{r=1}^{R} \boldsymbol{A}_{ir}\boldsymbol{B}_{jr}\boldsymbol{C}_{kr}\right)^2 \\ &= \min_{\boldsymbol{B}} \|\mathcal{T}^{(2)} - \boldsymbol{B}(\boldsymbol{C} \odot \boldsymbol{A})\|_F^2 \end{aligned} \tag{12}$$

Solving for $\boldsymbol{B}$, we get,

$$\boldsymbol{B} = \left(\mathcal{T}^{(2)}(\boldsymbol{C} \odot \boldsymbol{A})\right)(\boldsymbol{C}^T\boldsymbol{C} \circ \boldsymbol{A}^T\boldsymbol{A})^{-1} \tag{13}$$

Step 3: Likewise, keeping $\boldsymbol{A}, \boldsymbol{B}$ fixed, we can solve for $\boldsymbol{C}$ as

$$\begin{aligned} \mathcal{J}_C &= \min_{\boldsymbol{C}} \sum_{i,j,k} \left(\mathcal{T}_{i,j,k} - \sum_{r=1}^{R} \boldsymbol{A}_{ir}\boldsymbol{B}_{jr}\boldsymbol{C}_{kr}\right)^2 \\ &= \min_{\boldsymbol{C}} \|\mathcal{T}^{(3)} - \boldsymbol{C}(\boldsymbol{A} \odot \boldsymbol{B})\|_F^2 \end{aligned} \tag{14}$$

Solving for $\boldsymbol{C}$, we get,

$$\boldsymbol{C} = \left(\mathcal{T}^{(2)}(\boldsymbol{A} \odot \boldsymbol{B})\right)(\boldsymbol{A}^T\boldsymbol{A} \circ \boldsymbol{B}^T\boldsymbol{B})^{-1} \tag{15}$$

Further, after each step, we also normalize the columns of the determined factor matrix. After normalization of $\boldsymbol{C}$ following equation (15) we store the normalizing factors in a vector $\lambda \in \mathbb{R}^R$. From the above equations, the iterative CP-ALS algorithm can be formulated as follows

Initialize $\boldsymbol{B}, \boldsymbol{C}$;
**while** *not converged* **do**
  Solve for $\boldsymbol{A}$ using equation (11);
  Normalize Columns of $\boldsymbol{A}$;
  Solve for $\boldsymbol{B}$ using equation (13);
  Normalize Columns of $\boldsymbol{B}$;
  Solve for $\boldsymbol{C}$ using equation (15);
  Normalize Columns of $\boldsymbol{C}$, store it in $\lambda$;
  Calculate change in error or fitness $\delta$;
  **if** $\delta < tolerance$ **then**
    *converged=True*
  **end**
**end**

**Algorithm 1:** CP-ALS Algorithm

## METHODS

### Tensor Initialization

For our experiments, we generated synthetic tensors using the same strategy as described in Battaglino et al. [3]. We create tensor ($\mathcal{T}$) based on factor matrices ($\boldsymbol{A}, \boldsymbol{B}, \boldsymbol{C}$). This way, we would know the dependencies in the tensor data. We consider $3^{rd}$ order problem. The tensor is of size $n$ x $n$ x $n$. We used the true rank of matrices $\boldsymbol{A}, \boldsymbol{B}$ and $\boldsymbol{C}$ as $\frac{n}{2}$. Therefore, the size of factor matrices is $n$ x $\frac{n}{2}$ and all elements are real-values. The factor matrices are such that all the columns have the same colinearity $c$ (any two columns of a factor matrix would satisfy the equation given below).

$$c = \frac{a_r^{(i)T} a_r^{(j)}}{||a_r^{(i)T}|| \cdot ||a_r^{(j)}||} \tag{16}$$

To generate such a matrix, we followed the approach mentioned in Tomasi et al. [4]. High collinearity makes the recovery of the original factors difficult. Therefore, we fixed the colinearity to 0.62 in our experiments. This depends on the congruence factor given in Tomasi et al. [4]. We choose different congruence factors for each n (size of the tensor) to ensure that the resulting colinearity is fixed to be approximately 0.62.

$$\boldsymbol{A} = MR, \tag{17}$$

where $M$ is congruence matrix of size $\frac{n}{2} \times \frac{n}{2}$ such that the diagonals are 1 and all the rest of the values are set to congruence factor, while $R$ is a random matrix of size $n \times \frac{n}{2}$ with orthonormal column vectors.

**TABLE 1**. PARAMETERS

| n (size of tensor) | Congruence | Number of components |
|---|---|---|
| 50 | 0.18 | 80 |
| 100 | 0.14 | 120 |
| 200 | 0.105 | 180 |

$$\mathcal{T}_{i,j,k} = \sum_{f=1}^{n} \boldsymbol{A}_{i,f} \cdot \boldsymbol{B}_{j,f} \cdot \boldsymbol{C}_{k,f} \tag{18}$$

Then, we added noise to the tensor.

$$\mathcal{T} = \mathcal{T}_{true} + \eta \left( \frac{||\mathcal{T}_{true}||_F}{||\mathcal{K}||_F} \right) \mathcal{K}, \tag{19}$$

where $\eta = 0.05$ is the amount of noise and K is a tensor of equal size as T and the entries are drawn from a standard normal distribution.

### Implementing the CP-ALS Algorithm

We programmed the CP-ALS according to Algorithm 1 in C language, starting with a serial code. Details of the critical parts of the code are given below.

**Initialization** We initialized factor matrices with uniform random values between 0 and 1.

**MTTKRP** The first term of the products in equations (11), (13), and (15) involve the computation of the Khatri-Rao product, which is a highly memory-intensive step. However, certain structural features of the matrices can be exploited to make the computation of the term $\bar{A} = \left(\mathcal{T}^{(1)}(\boldsymbol{C} \odot \boldsymbol{B})\right)$ - called the Matrisized Tensor times Khatri-Rao Product (MTTKRP) a much simpler manner, as given in the equation below. This is inspired by Rollinger et al. [5].

$$\bar{A}_{i,r} = \sum_{j,k} \mathcal{T}_{i,j,k}\, \boldsymbol{B}_{j,r}\, \boldsymbol{C}_{k,r} \tag{20}$$

Likewise, the MTTKRP terms $\bar{B}$ and $\bar{C}$ can be computed similarly. This step is one of the most compute-intensive steps of

the code, whose complexity is $\mathcal{O}(Rd_1d_2d_3)$ floating-point operations, comprising of four for loops for each MTTKRP computation.

**Inverse** The second term of the products in equations (11), (13) and (15) involve the computation of the inverse of the term $\tilde{A}_{R\times R} = \boldsymbol{C}^T\boldsymbol{C} \circ \boldsymbol{B}^T\boldsymbol{B}$. Fortunately, $\tilde{A}$ would be a symmetric positive definite matrix, which would allow us to perform Cholesky Decomposition ($\tilde{A} = LL^T$) followed by determining the inverse of $L$ and thereby computing $\tilde{A}^{-1}$.

**Normalization** After the computation of each factor matrix as per equations (11), (12), and (13), we normalized its columns. We adopted the idea of MATLAB's implementation of CP-ALS (*www.tensortoolbox.org*) for normalizing, in which 2-norm is used in the first iteration, and $\infty$-norm is used for subsequent iterations. The normalization factors for $\boldsymbol{C}$ were stored in the array $\lambda$.

**Error and Fitness** The residual sum-of-squares error objective in equation (4) can be expressed as

$$e^2 = \langle \mathcal{T}, \mathcal{T} \rangle + \langle \hat{\mathcal{T}}, \hat{\mathcal{T}} \rangle - 2\langle \mathcal{T}, \hat{\mathcal{T}} \rangle \tag{21}$$

which gives,

$$e = \sqrt{\langle \mathcal{T}, \mathcal{T} \rangle + \langle \hat{\mathcal{T}}, \hat{\mathcal{T}} \rangle - 2\langle \mathcal{T}, \hat{\mathcal{T}} \rangle} \tag{22}$$

where,

$$\langle \mathcal{T}, \mathcal{T} \rangle = \|\mathcal{T}\|_F^2 \tag{23}$$

$$\langle \hat{\mathcal{T}}, \hat{\mathcal{T}} \rangle = \|\hat{\mathcal{T}}\|_F^2 = \lambda^T (\boldsymbol{A}^T\boldsymbol{A} \circ \boldsymbol{B}^T\boldsymbol{B} \circ \boldsymbol{C}^T\boldsymbol{C})\lambda \tag{24}$$

and

$$\langle \mathcal{T}, \hat{\mathcal{T}} \rangle = \sum_{r=1}^{R} \lambda_r \left( \sum_{i,j,k} \mathcal{T}_{i,j,k}\, \boldsymbol{A}_{i,r}\, \boldsymbol{B}_{j,r}\, \boldsymbol{C}_{k,r} \right) \tag{25}$$

The fit of the estimate $f$ can be determined as follows,

$$f = 1 - \frac{e}{\langle \mathcal{T}, \mathcal{T} \rangle} \tag{26}$$

### CPU-level Parallelization using OpenMP

We parallelized the code using OpenMP for a shared memory system. The first part of the code is the synthesis of tensor data and the second part is tensor decomposition using the CP-ALS method. To keep the generated matrix the same, we used the same random seed and avoided parallelization of any step that used a random number generator. This would ensure the replicability of our code.

### GPU-level Parallelization using OpenACC

We parallelized the code using OpenACC, which utilizes GPU. Similar to the OpenMP code, we parallelized the appropriate loops using the `#pragma acc parallel loop ...` command. To the best of our knowledge, we optimized for data transfer to avoid unnecessary communication between CPU and GPU.

## RESULTS

### Serial code

The tolerance required for convergence is set of 1e-8 for problem size of $n = 50$ and $n = 100$, while is is 1e-4 for problem size $n = 200$. The number of iterations for convergence for $n = 50$ is 2821 with the fit of 0.964, $n = 100$ is 1090 with the fit of 0.973 and $n = 200$ is 54 with the fit of 0.945.

### OpenMP

We parallelized the serial code for the shared memory system using OpenMP. It is seen that the Matricized Tensor times Khatri-Rao Product (MTTKRP) function is the most time-consuming step as it is of the order $O(n^4)$. After parallelizing the MTTKRP function, we saw the most significant increase in speed up. Other important contributors to the time are matrix multiplication, Hadamard product, transpose of a matrix, and tensor reconstruction. Most of these operations were easy to parallelize. These functions contribute to the maximum speed up. However, some matrix operations such as Cholesky decomposition, the inverse of a tridiagonal matrix, and Gram Schmidt orthogonalization did not have the same potential for parallelization due to data dependency.

Figure 1 shows the variation of speed up with the number of threads used for the tensor synthesis part. It is seen that speed up keeps increasing with an increase in the number of threads. However, as Gram Schmidt orthogonalization (required to generate the random column-orthonormal matrix) is not parallelized, the speed up remains low in the absolute sense. With the high number of threads (32), it is seen that there is a drastic decrease in the speed up. This may be due to the significant overhead associated with distributing the data across 32 threads such that it surpasses the benefits gained by the parallelization. However, such a trend is also seen with the larger matrix of size 200.

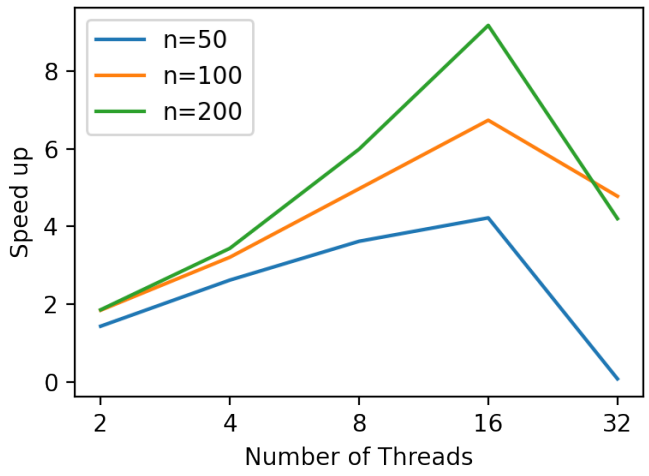


**FIGURE 1**. SPEED UP FOR TENSOR SYNTHESIS USING OPENMP

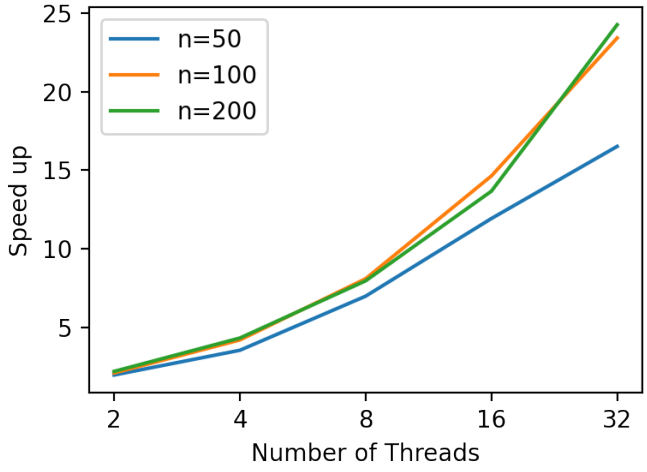


**FIGURE 2**. SPEED UP FOR CP-ALS TENSOR DECOMPOSITION USING OPENMP

Figure 2 shows the variation of speed up with the number of threads used for the CP-ALS decomposition part. The most time-consuming steps of CP-ALS decomposition are MTTKRP, the inverse of a symmetric matrix (which involves transpose, Cholesky decomposition, the inverse of a low triangular matrix, and matrix multiplication) and reconstructing tensor for error calculating. Most of these steps are parallelizable using OpenMP. Therefore, we see a significant increase in the speed up on increasing the number of cores. This speedup becomes higher as the problem size increases. This is clearly seen in the figure. The speedup for a small problem size and a large number of threads remains low due to the parallel overhead. The associated reduction of parallelization diminishes as the problem size becomes smaller as the non-parallelizable part becomes significantly time-consuming and the overhead associated with parallelization increases.

### OpenACC

Our code did not perform as expected due to some issues in data transfer mechanisms and produced a zero result every time. We wish to investigate the correctness of our code and rectify the errors.

## DRAWBACKS AND CHALLENGES

Although the alternating least squares approach for performing CP decomposition of a tensor is simple and easy to implement, there are a few notable drawbacks of this algorithm. First, the factorization is very sensitive to the starting values of the factor matrices. Second, the convergence of the CP-ALS algorithm is extremely slow and computationally expensive for large matrices, and there is no guarantee of reaching the global optimum. Hence, for huge sparse matrices, the CP-ALS algorithm is, in most cases, not the appropriate algorithm. We would also acknowledge certain drawbacks of our parallel algorithm. Parallelization was performed only at a higher level on most of the `for` loops. In other words, in areas where there are nested loops consisting of multiple `for` loops that can be parallelized, we chose to parallelize only the outermost loop in all cases. This way, full-fledged parallelism was not leveraged, and our speedups might not be optimal. Also, it is evident from the nature of the algorithm that distributed computation is highly favorable - for example, dividing the tensor or matrices into blocks and performing computation separately on different processors. This level of parallelization appears more efficient than using shared memory and is possible using MPI. We faced many challenges in data communication when we were implementing MPI, and we did not proceed with that further. Lastly, we did not try out input sizes beyond $n = 200$. Running the codes with large input sizes such as $n = 500, 1000$ would have better highlighted the scalability of our parallel approach.

## DISCUSSION AND FUTURE WORK

Tensor decomposition algorithms are rapidly evolving, and the newer algorithms are better in terms of speed, accuracy, scalability, robustness, and parallelizability. Our humble attempt to parallelize the simplest CP decomposition algorithm that uses an alternating least-squares approach has been successful with OpenMP. The choice of the rank $R$, i.e., the number of rank-1 components to restore the tensor, and the convergence criteria

influence the speed and accuracy of the reconstruction. Higher values of $R$ and stricter convergence criteria can improve the accuracy to a reasonable extent, but the speeds will be slower. However, we also need to take into account that if we keep increasing $R$ in the hope of improving the fit, there are chances that the model will overfit the noisy data, which is generally undesirable.

Tensor decomposition algorithms are, in general, highly parallelizable in many parts of the code, and detailed analysis must be performed to identify areas that can be parallelized in an optimized manner. As reported in other studies which attempted to parallelize the CP-ALS algorithm, using a combination of OpenMP and MPI with minimal data transfers is an extremely powerful approach, in addition to GPU-level parallelization. In the future, we would like to improve our parallel code and fully parallelize all possible sections. We will also work on correcting the data copying mechanism in our OpenACC code. Further, parallelizing other sophisticated variants of the CP algorithm, like CPRAND [6] and CP-ARLS-LEV [7] would be another interesting future direction. Lastly, we are very much looking forward to implementing our parallel algorithm on large real-world datasets to analyze the performances and to extract meaningful domain-specific information from the factor matrices. Prominent multidimensional datasets include GTEx (Genotype-Tissue Expression Dataset) [8] multi-tissue gene expression dataset and social media datasets (e.g. Reddit and Twitter).

## CONCLUSION

The recent decade witnessed an explosion in the magnitude of data getting generated. Higher-order tensors are becoming increasingly popular for data storage and processing. In this regard, specialized algorithms that are suitable for tensors are required. Low-rank Tensor decomposition helps in breaking down tensors into meaningful components and denoising the data. In this project, we have attempted to parallelize the CP-ALS (alternating least-squares) tensor decomposition algorithm using OpenMP and OpenACC. While we were successful in CPU-level parallelization using OpenMP, we faced issues while implementing OpenACC due to possibly incorrect data copying mechanisms, which we look forward to correcting it. Overall, we have highlighted the performance and parallelizability of the CP-ALS algorithm.

## CODE AVAILABILITY

The codes are also available on the given github repository. *https://github.com/BurhanSabuwala/PSC_project*.

## ACKNOWLEDGMENT

We are incredibly thankful and grateful towards our Professors, Dr. Kameswararao Anupindi and Dr. Rupesh Nasre for their continual support and guidance while offering the ID5130 course. Their lectures were fascinating and enhanced our knowledge to a great extent. We would also like to thank the people responsible for the smooth management of the Aqua cluster, thereby helping us complete our assignments and project. We want to express our gratitude to our classmates of the ID5130 course for offering help and being supportive. Lastly, we were very much inspired by Dr. Tamara Kolda's lecture videos and publications on Tensor Decomposition.

## REFERENCES

[1] Rabanser, S., Shchur, O., and Günnemann, S., 2017. "Introduction to tensor decompositions and their applications in machine learning". *arXiv preprint arXiv:1711.10781*.

[2] Mahyari, A. G., Zoltowski, D. M., Bernat, E. M., and Aviyente, S., 2016. "A tensor decomposition-based approach for detecting dynamic network states from eeg". *IEEE Transactions on Biomedical Engineering,* ***64***(1), pp. 225–237.

[3] Battaglino, C., Ballard, G., and Kolda, T. G., 2018. "A practical randomized cp tensor decomposition". *SIAM Journal on Matrix Analysis and Applications,* ***39***(2), Jan, p. 876–901.

[4] Tomasi, G., and Bro, R., 2006. "A comparison of algorithms for fitting the parafac model". *Computational Statistics Data Analysis,* ***50***(7), pp. 1700–1734.

[5] Rolinger, T. B., Simon, T. A., and Krieger, C. D., 2017. "Performance challenges for heterogeneous distributed tensor decompositions". In 2017 IEEE High Performance Extreme Computing Conference (HPEC), pp. 1–7.

[6] Battaglino, C., Ballard, G., and Kolda, T. G., 2018. "A practical randomized cp tensor decomposition". *SIAM Journal on Matrix Analysis and Applications,* ***39***(2), pp. 876–901.

[7] Larsen, B. W., and Kolda, T. G., 2020. "Practical leverage-based sampling for low-rank tensor decomposition". *arXiv preprint arXiv:2006.16438*.

[8] Lonsdale, J., Thomas, J., Salvatore, M., Phillips, R., Lo, E., Shad, S., Hasz, R., Walters, G., Garcia, F., Young, N., et al., 2013. "The genotype-tissue expression (gtex) project". *Nature genetics,* ***45***(6), pp. 580–585.