A Parallel CP Decomposition Algorithm for Sparse Tensor

Zhi-cheng Ma\textsuperscript{1,a}, Yu Wu\textsuperscript{2,b}, Guang-hui Yan\textsuperscript{2,c,*}, Ya-fei WANG\textsuperscript{2} and Qing-qing Ma\textsuperscript{2}

\textsuperscript{1}Gansu Shining Science & Technology Co., Ltd., Lanzhou, Gansu, China
\textsuperscript{2}Lanzhou Jiaotong University, Lanzhou, Gansu, China
\textsuperscript{a}mac3163@yahoo.com.cn, \textsuperscript{b}wuyu1028r@163.com, \textsuperscript{c}yan\_guanghui@163.com

*Corresponding author

**Keywords:** CP Decomposition, CP-ALS, GPU, CUDA.

**Abstract.** With the advent of high-dimensional data, tensor and tensor decomposition methods have received widespread attention in the field of data analysis. However, the high-dimensional and sparse characteristics of tensor data lead to the high computation complexity of algorithm, which became an obstacle to the actual application of the tensor decomposition algorithm. In this paper, we start from the high sparsity of real tensor, and give a GPU parallel sparse tensor CP decomposition algorithm called ParSCP-ALS.

**Introduction**

In the past ten years, the tensor and tensor decomposition have become popular in high-dimensional data analysis. The tensor decomposition method is similar to matrix decomposition, which can map high-dimensional data to a lower subspace, while reducing the dimension of data. So far, tensor and tensor decomposition have applied in many fields, such as signal processing, computer vision, data mining, graph analysis, etc.[1]

Due to the high-dimensional nature of tensor, all tensor decomposition algorithms have high computation complexity. At the same time, the high-dimensional data, such as DBLP[2] and Movielens dataset[3], are always sparse, and the size of data grows exponentially with the dimension. The efficiency of algorithm and sparseness of large-scale data prevent tensor decomposition for further application in practical.

To solve these problems of tensor decomposition, many researchers have made several improvements. Papalexakis et al [4] proposed the ParCube algorithm and Antikainen et al [5] speed up the decomposition of non-negative tensors with GPU. The existing research results show that the GPU parallel computing method can effectively accelerate the tensor decomposition algorithm. In this paper, aiming at the high sparsity of tensor data, which is the most significant feature of real high-dimensional data, we proposed an algorithm called ParSCP-ALS, which improved the CP-ALS algorithm for CP decomposition and use GPU to accelerate the algorithm at the same time.

The rest of this paper is organized as follow. Section 2 reviews the CP decomposition and CP-ALS algorithm. Section 3 introduces the proposed ParSCP-ALS algorithm, including the improvement of the traditional algorithm, the data structure for sparse data and the parallelization on GPU. Section 4 presents the experiment result on DBLP and MovieLens dataset.

**Background Knowledge**

**Tensor and matricization**

A tensor is a mathematical representation of multi-way arrays. The dimension of tensor is defined as order or mode. In this paper, we use $X$ as a tensor, $A$ as a matrix, and $x$ as a vector. The
Matricization is a process of reordering the elements of a tensor into a matrix. A N-dimension tensor has N ways of matricization, which corresponds to the N matrix. The n-th matricization of a tensor $X \in \mathbb{R}^{I \times J \times K}$ is defined as mode-n matrix $X_{(n)}$. After reordering as a matrix, a tensor can product another matrix, which called the mode-n product of tensor. The mode-n product of tensor $X \in \mathbb{R}^{I \times J \times K}$ with a matrix $U \in \mathbb{R}^{I \times J}$ is donated by $Y = X \times_n U \in \mathbb{R}^{I \times J \times K}$.

### CP Decomposition and CP-ALS Algorithm

Tensor decomposition can be regarded as a form of higher-order principle component analysis. The CP decomposition factorizes a tensor into a sum of component rank-one tensors. For example, given a third-order tensor $X \in \mathbb{R}^{I \times J \times K}$, we wish to write it as

$$X \approx \sum_{r=1}^{R} a_r \circ b_r \circ c_r,$$

where $R$ is a positive integer and $a_r \in \mathbb{R}^I$, $b_r \in \mathbb{R}^J$, $c_r \in \mathbb{R}^K$, for $r = 1, ..., R$. Otherwise, it can be write as

$$X \approx \sum_{r=1}^{R} \lambda_r a_r \circ b_r \circ c_r.$$

The factor matrix refer to combination of vectors from the rank-one components, i.e., $A = [a_1, a_2, ..., a_R] \in \mathbb{R}^{I \times R}$ and likewise for $B$ and $C$.

![Figure 1. The CP Tensor Decomposition of a third order tensor.](image)

The CP-ALS algorithm is the alternating least squares (ALS) method for CP decomposition, which was proposed in the original papers by Carrol and Chang. The ALS algorithm fixe two factor-matrixs to solve the other one, and continue to repeat the entire procedure until the convergence criterion is satisfied.

**Algorithm 1 CP-ALS**

**Input:** Tensor $X \in \mathbb{R}^{I \times J \times K}$, Rank $R$, MaxIteration $T$, Factor Matrix $A, B, C$;

**For** $t = 1, ..., T$ **do**

1. $A = X_{(i)} \left( C \circ B \right) \left( C^T C \ast B^T B \right)^{\dagger}$, Normalize $A$, $\lambda = \text{norms}$;
2. $B = X_{(j)} \left( C \circ A \right) \left( C^T C \ast A^T A \right)^{\dagger}$, Normalize $B$, $\lambda = \text{norms}$;
3. $C = X_{(k)} \left( B \circ A \right) \left( B^T B \ast A^T A \right)^{\dagger}$, Normalize $C$, $\lambda = \text{norms}$;

**if** convergence criterion $\varepsilon$ is met **do**

```
break the loop;
```

**end if**

**end for**

**Output:** $A \in \mathbb{R}^{I \times R}$, $B \in \mathbb{R}^{J \times R}$, $C \in \mathbb{R}^{K \times R}$, $\lambda \in \mathbb{R}^R$.

For ease of presentation, we only derive the algorithm in the third-order case, and we donated the Kronecker product by $\otimes$, the Khatri-Rao product by $\odot$ and the Hadamard product by $\ast$. The factor matrix $A, B$ and $C$ can be initialized with the first $R$ left singular vectors of mode-n matrix $X_{(i)}$. 

94
The Parallel Implementation

Tensor Khatri-Rao Product

In the CP-ALS algorithm, computing \( A = X_{(i)} (C \odot B) \left( C^T C \ast B^T B \right)^T \) has a high computation cost and a lengthy processing time. To overcome this deficiency, we give a simplified method to calculate the step of \( X_{(i)} (C \odot B) \).

We define it as Tensor Khatri-Rao product (TKR). The TKR product aim to use the property of Khatri-Rao product to convert the calculation process equivalently, and get the calculation result directly without generating the intermediate variable, so as to avoid redundant calculation and storage. Given the mode-1 matrix \( X_{(i)} \in \mathbb{R}^{I \times JK} \), \( B \in \mathbb{R}^{J \times R} \), \( C \in \mathbb{R}^{K \times R} \), so the TKR product can be expressed as

\[
M_i = X_{(i)} (C \odot B) \in \mathbb{R}^{I \times R}.
\]

Suppose \( Y = (C \odot B) \in \mathbb{R}^{JK \times R} \), and elementwise, \( Y (i, r) = C \left( \lfloor j/J \rfloor , r \right) \times B \left( 1 + (i-1) \% J, r \right) \).

Sparse Data Structure CSR

The Compressed Sparse Row (CSR) format divide and compress the matrix into rows and store the non-zero elements in order.

\[
\begin{array}{cccc}
1 & 7 & 0 & 0 \\
0 & 2 & 8 & 0 \\
5 & 0 & 3 & 9 \\
0 & 6 & 0 & 4 \\
\end{array}
\]

\[
\begin{array}{c}
[0, 2, 4, 7, 9] \rightarrow \text{RowPtr} \\
[0, 1, 1, 2, 0, 2, 3, 1, 3] \rightarrow \text{Col} \\
[1, 7, 2, 8, 5, 3, 9, 6, 4] \rightarrow \text{Val}
\end{array}
\]

Figure 2. The Example of CSR.

We can simply use CSR format to store the spare data during computing. After using CSR format, the number of nonzero elements in i-th row is defined as \( N_i \), i.e., \( N_i = \text{RowPtr}_i - \text{RowPtr}_{i-1} \), and the TKR product can be represent as

\[
M (i, r) = \sum_{j=1}^{JK} \left[ X_{(i)} (i, j) \times C \left( \lfloor j/J \rfloor , r \right) \times B \left( 1 + (i-1) \% J, r \right) \right].
\]

ParSCP-ALS Algorithm

We briefly review the features of GPU programming platform called CUDA. The program on CUDA consists of several kernel functions, which are suitable for parallel execution on GPU. Each kernel function can executed by each thread block, and each block contains a fixed number of threads. The threads per block can be synchronized with each other and have a high-speed shared memory for inter-thread communication[6].

To make our parallel implementation of TKR product more effective, we should divide more threads and let the computing task for each thread minimum. Firstly, we divide GPU into \( N \) blocks and \( R \) threads per block, where \( N \) and \( R \) is the row and column number of matrix \( M \). It means that there are \( N \times R \) threads in total and each element in \( M \) corresponds to a single thread. Then, to avoid the conflict of data usage between different thread, we use the shared memory in per block to store the values of each row of \( X_{(a)} \).
Algorithm 2 TKR kernel

Input: $X^{(i)} \in \mathbb{R}^{I \times J \times K}$, $B \in \mathbb{R}^{J \times R}$, $C \in \mathbb{R}^{K \times R}$, Rank $R$

Set $bid = blockIdx.id$, $tid = threadIdx.id$, allocate SharedMemory $smCol$ and $smVal$ for bidth row of $X^{(i)}$;

Set $N_i$ as the number of nonzero elements of bidth row of $X^{(i)}$;

$Ni = RowPtr_{bid+1} - RowPtr_{bid}$;

for $n = 1 : N_i$ do

Copy $Col_{RowPtr_{bid} + n}$ to $smCol_{n}$, $Val_{RowPtr_{bid} + n}$ to $smVal_{n}$;

end for

for $i = 1 : N_i$ do

$m_{bidid} = \sum_{j} \sum_{k} smVal_{j} \times C_{smCol_{j}} \times B_{k}$;

end for

Free $smCol$, $smVal$;

Output: $M_i \in \mathbb{R}^{J \times R}$

The kernel function of TKR product is showed in Algorithm 2. The Flow Chart of parallel TKR product is showed in Fig.3. We define the tensor decomposition algorithm with CSR sparse format and TKR product as Sparse CP-ALS (SCP-ALS), and define the algorithm with parallel TKR product as Parallel SCP-ALS (ParSCP-ALS). In ParSCP-ALS, we have $A = M_1 M_2$, $M_1 = TKR\left(X^{(i)}, B, C\right)$, $M_2 = \left(C^T C * B^T B\right)^{+}$.

Figure 3. The Flow Chart of the Parallel TKR Product.

The Experiments

Our experiments are performed on an Intel i7-6700K CPU, 32G RAM and NVIDIA GeForce GTX1070 platform, and the system is win10 64bit and CUDA 8.0. For the sake of simplicity, we use the svds() function in MATLAB to compute the singular vectors of mode-n matrix $X^{(i)}$ during initialization, and let Rank $R = 10$, Max iteration $T = 50$ and convergence criterion $\varepsilon = 10^{-6}$.

We use the Movielens and DBLP dataset to evaluate the performance of our parallel algorithm. DBLP is a computer science bibliography website, which list millions of journal articles, conference papers, and other publications on computer science. The DBLP dataset contains information such as author, title, years and article type for each publication. We build high-dimensional data using collaborations between authors who co-authored articles in the same journal, and represent as (author-author-journal). In the data of year ‘2016’ and publication type ‘article’, we randomly selected 5 groups of data.
Table 1. The Information of DBLP Datasets.

<table>
<thead>
<tr>
<th>DBLP</th>
<th>Article</th>
<th>Author(I,J)</th>
<th>Journal(K)</th>
<th>nonzero</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1</td>
<td>1K</td>
<td>3888</td>
<td>415</td>
<td>6790</td>
</tr>
<tr>
<td>D2</td>
<td>2.5K</td>
<td>9645</td>
<td>655</td>
<td>18756</td>
</tr>
<tr>
<td>D3</td>
<td>5K</td>
<td>16607</td>
<td>845</td>
<td>35941</td>
</tr>
<tr>
<td>D4</td>
<td>7.5K</td>
<td>25010</td>
<td>960</td>
<td>49104</td>
</tr>
<tr>
<td>D5</td>
<td>10K</td>
<td>32254</td>
<td>1046</td>
<td>67263</td>
</tr>
</tbody>
</table>

Movielens is a web-based recommender that recommends movie for its users to watch. The dataset contains the rating data collected from website, such as users, movies, tag and rating. We build the high-dimensional data from the user rating for movies with same tag, which represent as (user-movie-tag). The data contains 3884 movies, 6040 users, 18 movie tags, and 2101816 rating data in total.

In experiments, we compare ParSCP-ALS with SCP-ALS, ParCP-ALS and CP-ALS. The ParCP-ALS is a representation of existing parallelization for tensor decomposition, which directly use GPU to compute the matrix product in parallel.

![Figure. The Performance Comparisons on DBLP Datasets and Movielens Dataset.](image)

Fig 4.1 shows the runtime of ParSCP-ALS, SCP-ALS and ParCP-ALS on DBLP datasets. The runtime of CP-ALS on D1 and D2 is 35s and 846s, which lags behind that of other algorithm, and are not showed in the figure. When the CP-ALS algorithm runs on the D3, D4 and D5 on DBLP, the memory required for computation exceeds the actual memory, and the result cannot be calculated.

Fig 4.2 showed the runtime of ParSCP-ALS and other algorithms on the Movielens dataset. We define the speedup ratio as $s = T_s/T_p$, where $T_s$ is the runtime of serial algorithm and $T_p$ is the runtime of parallel algorithm. From the experiment result, we know that the speedup between ParSCP-ALS and SCP-ALS is 1.8, and the speedup ratio between ParCP-ALS and CP-ALS is 2.6.

Summary

Our proposed parallel CP tensor decomposition algorithm ParSCP-ALS perform better than other existing algorithms in real sparse data experiments. And the ParSCP-ALS algorithm is able to facing the data of same size but different sparsity, such as dataset of DBLP D1 and Movielens.

Acknowledgements

This work is supported by National Natural Science Foundation under Grant No.61363060 No.61662066 and No. 61163010, the Science project of Gansu Electric Power Information & Communication Centre. This work is sponsored by the Science Project of Gansu Shining Science &Technology Co., Ltd.

References