结合 GPU 技术的并行 CP 张量分解算法

武 昱 闫光辉 王雅斐 马青青 刘宇轩
(兰州交通大学电子与信息工程学院 兰州 730030)

摘 要 随着高维数据的涌现,张量和张量分解方法在数据分析领域中受到了广泛关注。然而,张量数据的高维度和稀疏特性,导致算法的复杂度较高,阻碍了张量分解算法在实际中的应用。许多学者通过引入并行计算来提升张量分解算法的计算效率。在现有研究的基础上,给出一种简化计算 Khatri-Rao 乘积的 GPU 并行 CP 张量分解算法,称为 ParSCP-ALS。在模拟数据集和真实数据集上的实验结果显示,相比现有并行算法,文中设计的 ParSCP-ALS 算法能有效提高 CP 张量分解的计算效率,其中在 Movielens 数据集上的计算时间减少了约 58%。

关键词 CP 张量分解,CP-ALS 算法,GPU,CUDA

中图法分类号 TP391 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2018.11.048

Parallel CP Tensor Decomposition Algorithm Combining with GPU Technology

WU Yu YAN Guang-hui WANG Ya-fei MA Qing-qing LIU Yu-xuan
(School of Electronic and Information Engineering, Lanzhou Jiaotong University, Lanzhou 730030, China)

Abstract With the emergence of high-dimensional data, tensor and tensor decomposition have draw widespread attention in the field of data alanlysis. The high dimensionality and sparsity of tensor data results in a high computational complexity of tensor methods, which becomes an obstacle to the application of tensor decomposition in practical. Many researchers have introduced the parallel computing methods to improve the efficiency of tensor decomposition algorithm. Based on the existing research, this paper presented a GPU parallel CP tensor decomposition algorithm through simply calculating Khatri-Rao product, called ParSCP-ALS algorithm. The experimental results show that the proposed ParSCP-ALS algorithm can effectively improve the computational efficiency of CP tensor decomposition compared with the existing parallel algorithm. On the Movielens data set, the ParSCP-ALS algorithm reduces the computational time by about 58% compared with the existing parallel algorithm.

Keywords CP tensor decomposition, CP-ALS, GPU, CUDA

1 引言

计算机技术和网络技术的快速发展使得越来越多实体间的相互联系都可被记录,例如社交网络用户之间的好友关系、互联网页面之间的链接关系等[1]。对于包含二元关系的图,学者们已经得到了大量的研究成果和挖掘方法[2],但大部分现有的研究工作都是面向简单的图,而现实中多个实体之间的复杂关系是难以用简单图表示与分析的,例如社交网站中用户对其他用户关于某个事件的评论,电影评分网站中用户对不同类型的电影的评分,城市中公交和地铁等不同类型的交通网络之间的连接关系等[1]。这些复杂的关系需要通过高维的数据形式进行表示和处理,如何处理这些高维数据成为了国内外研究者关注的重点。在近十年中,张量和张量分解方法成为了高维数据的常用处理方法[3]。由于张量的数据形式能够自然保留高维数据的多维度关系与结构特点,同时张量分解方法与矩阵分解类似,能将高维数据映射到低维空间,达到对数据压缩和降维的目的,因此张量和张量分解的分析方法已被应用到众多领域中,如信号处理、计算机视觉、数据挖掘、图分析等[3]。

张量分解的研究最早始于 1927 年,Hitchcock 等[4]提出了对张量进行分解的思想,用低秩分解(lower-rank decomposition)去近似原始张量。基于此,后续产生了很多种分解方法[5],如 CP 分解、Tucker 分解等,本文主要考虑 CP 张量分解。CP 张量分解[6]是 Harshman 等提出的平行因子分解(PARAFAC)与 Carroll 等[7]提出的张量的规范分解(Canonical Decomposition, CANDECOMP)两种等价模型的统称。当前计算 CP 张量分解的最常用的方法是交替最小二乘法(Alternating Least Squares, ALS),简称 CP-ALS 算法[7]。

CP 张量分解作为处理高维数据的特有方法,因张量数据自身的高维度特性与张量的相关运算,其对应算法具有较高

到稿日期:2017-10-20 返修日期:2018-01-26 本文受国家自然科学基金项目(61662066,61163010)资助。

武 昱(1993-),男,硕士,主要研究方向为张量分解、并行计算;**闫光辉**(1970-),男,教授,博士生导师,CCF 会员,主要研究方向为数据挖掘、复杂网络、社交网络等,E-mail:yan_guanghui@163.com(通信作者);**王雅斐**(1992-),女,硕士,主要研究方向为社区发现、节点重要性;**马青青**(1993-),女,硕士,主要研究方向为社区发现、链路预测;**刘宇轩**(1992-),男,硕士,主要研究方向为社区发现、离群点检测。

的时间复杂度和空间复杂度。为解决张量分解的算法在真实数据上实效性较差的问题,学者们在传统的 CP-ALS 算法上进行了许多改进。针对真实张量数据高稀疏性的特点,Papalexakis 等[8]提出了 ParCube 算法,即对原稀疏张量进行均匀采样,每次对一个规模很小的张量进行分解,最后通过合并得到原始张量的分解结果。基于并行计算技术,Antikainen 等[9]通过 GPU 并行的方式对非负张量的分解进行加速,也就是在 CP-ALS 算法的迭代过程中让每个 GPU 的线程块并行计算张量的每一个切片,从而并行得到最终的分解结果。Zou 等[10]提出了基于上下文推荐的并行张量分解算法——GPUTENSOR,并给出了 Tucker 张量分解在 GPU 上的并行算法。Kang 等[11]提出了基于 Hadoop 分布式平台的 GigaTensor 算法,给出了 PARAFAC 张量分解算法的并行实现,并利用 Hadoop 分布式结构存储量大的优点给出计算大规模 CP 张量分解的可行方案。

对现有改进算法的研究说明,充分利用数据的稀疏性特性和并行计算技术能够有效提高 CP 张量分解算法的计算效率。采用 GPU 并行的改进算法的计算效率提升显著[9-10],但算法并行化过程较复杂;采用 Hadoop 并行的改进算法[11]能够扩大所处理的张量分解问题的规模,解决单个计算机内存有限的问题,但消耗资源多且运行时间长[5]。

本文选取 GPU 并行计算作为提高张量分解算法效率的主要方式,因为张量分解的算法包含大量矩阵运算,而矩阵运算的特点为计算规模庞大、单个计算任务相对独立且简单。GPU 作为包含众多处理核心硬件的图形处理器,在面对计算大量数据的单指令多数据流(SIMD)的计算任务时,其处理性能远胜于 CPU[12]。而张量分解算法中许多计算任务的特性都符合 SIMD 的条件,因此 GPU 更适合本算法的并行实现[13]。

现有的采用 GPU 并行的算法只是将原张量分解算法并行实现[10],或是将原问题细分为小规模问题进行多线程并行分解[9],并未对原分解算法进行改进和优化。本文提出的结合 GPU 技术的并行 CP 张量分解算法,首先省去了冗余的计算步骤,降低了算法复杂度;然后针对真实张量数据高稀疏性的特点,选取稀疏的存储结构,节省了内存空间;最后结合 GPU 并行的特点给出相应的改进算法的并行实现。

本文的贡献如下:

1)针对 CP-ALS 算法中含有 Khatri-Rao 乘积的步骤计算复杂的问题,省去冗余的中间变量的计算消耗,给出简化的计算步骤,并证明简化方法与原方法的计算结果一致。

2)针对真实张量数据的高稀疏性特点,采用稀疏的数据结构 CSR 存储稀疏矩阵,减少计算中内存的占用,扩大了所能处理问题的规模;同时,稀疏的数据结构提高了并行矩阵的运行效率。

3)根据改进后的算法,结合 GPU 特点,给出并行程度高、并行任务小的并行 CP 张量分解算法——ParSCP-ALS 算法。

4)在模拟数据集上,验证了本文所提并行算法的有效性;在真实数据集上,对比测试了算法的实际性能,证明了本文提出的并行算法能更快地计算 CP 张量分解。

2　基础概念

2.1　张量

张量(tensor)是高维数组的总称。简单来说,一维数组称为向量,二维数组称为矩阵,三维及高维数组则称为高阶张量。

图 1 为一个三阶张量的示意图。

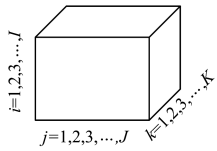


图 1　三阶张量 $\mathcal{X}\in\mathbb{R}^{I\times J\times K}$ 图例

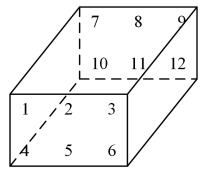
Fig. 1　Sample of third order tensor $\mathcal{X}\in\mathbb{R}^{I\times J\times K}$

2.2　张量的矩阵展开和模-n 乘积

张量的矩阵展开(matricization)[5]形式被称为张量的模-n 矩阵,是将张量按照不同的维度重新组合为矩阵。

定义 1　N 阶张量 $\mathcal{X}\in\mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ 在第 n 维展开的结果表示为 $X_{(n)}\in\mathbb{R}^{I_n\times(I_1\cdots I_{n-1}I_{n+1}\cdots I_N)}$。

图 2 以一个三阶张量为例,给出了其 3 个模-n 矩阵的展开形式。



$$X_{(1)}=\begin{bmatrix}1&2&3&7&8&9\\4&5&6&10&11&12\end{bmatrix}\quad X_{(2)}=\begin{bmatrix}1&4&7&10\\2&5&8&11\\3&6&9&12\end{bmatrix}$$

$$X_{(3)}=\begin{bmatrix}1&4&2&5&3&6\\7&10&8&11&9&12\end{bmatrix}$$

图 2　张量 $\mathcal{X}\in\mathbb{R}^{2\times3\times2}$ 的模-n 矩阵展开

Fig. 2　Matricization of tensor $\mathcal{X}\in\mathbb{R}^{2\times3\times2}$

张量的模-n 乘积表示的是张量与矩阵的乘积,其结果仍为张量[5]。

定义 2　N 阶张量 $\mathcal{X}\in\mathbb{R}^{I_1\times I_2\times\cdots\times I_N}$ 与矩阵 $U\in\mathbb{R}^{J\times I_n}$ 的模-n 乘积表示为:

$$\mathcal{Y}=\mathcal{X}\times_n U \tag{1}$$

其中,$\mathcal{Y}\in\mathbb{R}^{I_n\times(I_1\cdots I_{n-1}JI_{n+1}\cdots I_N)}$,也可以表示为矩阵形式:

$$Y_{(n)}=UX_{(n)} \tag{2}$$

2.3　CP 张量分解

CP 分解(Candecomp/Parafac)的主要思想就是将一个张量分解为有限个秩一张量(rank-one tensor)的和[5]。

图 3 为三阶张量的 CP 分解示意图。

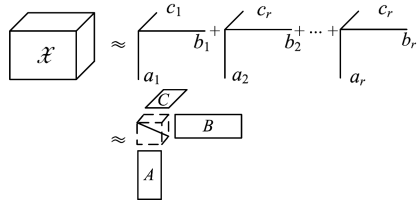


图 3　三阶张量的 CP 分解图例

Fig. 3　CP decomposition of third order tensor

三阶张量 $\mathcal{X}\in\mathbb{R}^{I\times J\times K}$ 的分解表示为：

$$\mathcal{X}\approx\sum_{r=1}^{R}a_r\circ b_r\circ c_r \tag{3}$$

其中，$\circ$ 表示秩一张量的外积，R 为张量的秩，$a_r\in\mathbb{R}^I$，$b_r\in\mathbb{R}^J$，$c_r\in\mathbb{R}^K$，$r=1,\cdots,R$。

或表示为：

$$x_{ijk}\approx\sum_{r=1}^{R}a_{ir}b_{jr}c_{kr} \tag{4}$$

将式(6)改为矩阵形式，可表示为：

$$\mathcal{X}\approx\sum_{r=1}^{R}\lambda_r a_r\circ b_r\circ c_r=[\![\lambda;A,B,C]\!] \tag{5}$$

其中，因子矩阵 $A=[a_1,a_2,\cdots,a_R]\in\mathbb{R}^{I\times R}$，$B\in\mathbb{R}^{J\times R}$，$C\in\mathbb{R}^{K\times R}$，$\lambda\in\mathbb{R}^R$ 为归一化系数[5]。

2.4 矩阵运算

本节给出在张量分解的算法中用到的矩阵运算：Kroncker 乘积，Khatri-Rao 乘积和 Hadamard 乘积[5]。本文中矩阵 A 的第 i 行第 j 列的元素表示为 a_{ij}，矩阵 A 的第 i 列表示为 $A_{(:,j)}$ 或者 a_j，向量 v 中的第 n 个元素表示为 v_n。

定义 3 矩阵 $A\in\mathbb{R}^{I\times J}$ 和 $B\in\mathbb{R}^{K\times L}$ 的 Kroncker 乘积表示为 $A\otimes B$，且 $A\otimes B\in\mathbb{R}^{IK\times JL}$。

$$A\otimes B=\begin{bmatrix}a_{11}B&\cdots&a_{1J}B\\ \vdots&\ddots&\vdots\\ a_{I1}B&\cdots&a_{IJ}B\end{bmatrix} \tag{6}$$

定义 4 矩阵 $A\in\mathbb{R}^{I\times J}$ 和 $B\in\mathbb{R}^{J\times K}$ 的 Khatri-Rao 乘积表示为 $A\odot B$，且 $A\odot B\in\mathbb{R}^{IJ\times K}$。

$$\begin{aligned}A\odot B&=[A_{(:,1)}\otimes B_{(:,1)}\cdots A_{(:,K)}\otimes B_{(:,K)}]\\&=[a_1\otimes b_1\cdots a_K\otimes b_K]\end{aligned} \tag{7}$$

定义 5 矩阵 $A\in\mathbb{R}^{I\times J}$ 和 $B\in\mathbb{R}^{J\times K}$ 的 Hadamard 乘积表示为 $A*B$，且 $A*B\in\mathbb{R}^{I\times J}$，表示为：

$$A*B=\begin{bmatrix}a_{11}b_{11}&\cdots&a_{1J}b_{1J}\\ \vdots&\ddots&\vdots\\ a_{I1}b_{I1}&\cdots&a_{IJ}b_{IJ}\end{bmatrix} \tag{8}$$

矩阵 X 的奇异值分解定义为 $X=U\Sigma V^{\mathrm{T}}$，矩阵 X 的伪逆定义为 $X^{\dagger}=V\Sigma^{-1}U^{\mathrm{T}}$。

3 结合 GPU 技术的 CP 张量分解算法

3.1 CP-ALS 算法

本节以三阶张量为例，给出计算 CP 张量分解最常用的 CP-ALS 算法的流程与算法复杂度分析。CP-ALS 算法的主要思想是在 3 个因子矩阵中选 2 个作为已知来更新另一个因子矩阵，重复该步骤，直到满足收敛条件时算法终止[5]。算法 1 给出了 CP-ALS 算法的伪代码。

算法 1 CP-ALS 算法

Input: Tensor $\mathcal{X}\in\mathbb{R}^{I\times J\times K}$, rank R, MaxIteration T, initialize A, B, C

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

1. for t = 1,…,T
2. $A=X_{(1)}(C\odot B)(C^{\mathrm{T}}C*B^{\mathrm{T}}B)^{\dagger}$;
3. normalize A, λ = norms;
4. $B=X_{(2)}(C\odot A)(C^{\mathrm{T}}C*A^{\mathrm{T}}A)^{\dagger}$;
5. normalize B, λ = norms;
6. $C=X_{(3)}(B\odot A)(B^{\mathrm{T}}B*A^{\mathrm{T}}A)^{\dagger}$;
7. normalize C, λ = norms;
8. if convergence criterion ε is met
9. break for loop;
10. end if
11. end for
12. return A, B, C, λ.

在 CP-ALS 算法初始化的过程中，因子矩阵 A,B,C 分别由模-n 矩阵 $X_{(n)}$ 的奇异值分解得到的左奇异值矩阵中的前 R 个奇异值向量组成。算法的收敛条件是两次迭代的因子矩阵变化很小或者达到最大迭代次数[7]。假设算法当前迭代得到的因子矩阵为 $\hat{A},\hat{B},\hat{C}$，若 $\|\hat{A}-\hat{A}_{\mathrm{old}}\|_F<\varepsilon$，则认为因子矩阵变化较小，满足算法的收敛条件。

下面对 CP-ALS 算法的复杂度进行分析。假设张量 $\mathcal{X}\in\mathbb{R}^{n\times n\times n}$，可知 $B\in\mathbb{R}^{n\times R}$，$C\in\mathbb{R}^{n\times R}$，则计算 $C\odot B$ 的时间复杂度约为 $O(n^2R)$；已知 $(C^{\mathrm{T}}C*B^{\mathrm{T}}B)\in\mathbb{R}^{R\times R}$，则计算 $(C^{\mathrm{T}}C*B^{\mathrm{T}}B$ 的时间复杂度约为 $O(2nR^2+R^2)$；已知 $X_{(1)}\in\mathbb{R}^{n\times n^2}$ 和 $C\odot B\in\mathbb{R}^{n^2\times R}$，则计算 $X_{(1)}$ 与 $(C\odot B)$ 的乘积的计算复杂度约为 $O(n^3R)$。在计算中秩 R 取较小的值，则 CP-ALS 算法的时间复杂度约为 $O(n^3R)$。在 CP-ALS 算法中，需要存储模-n 矩阵 $X_{(n)}$，因子矩阵 A,B,C 与中间变量 $(C^{\mathrm{T}}C*B^{\mathrm{T}}B)$ 和 $C\odot B$，则空间复杂度约为 $O(n^3)$。

CP-ALS 算法的时间复杂度和空间复杂度高的主要原因是计算过程中需要存储和计算大量的矩阵运算。现有的 GPU 并行技术能够在矩阵乘法计算中获得非常好的加速效果，Antikainen 等[9] 和 Zou 等[10] 将其应用到张量分解算法中，取得了不错的效果。

然而，现有算法[9-10]都是将原始算法并行实现，没有事先对算法进行改进，算法的总计算量在并行前后没有变化。为了进一步提升并行张量分解算法的性能，本文先改进算法，降低了稀疏情况下 CP 张量分解算法的复杂度，之后并行实现 CP-ALS 算法。在 CP-ALS 算法的步骤 2 中，若按照计算顺序依次计算矩阵乘法，会产生中间变量 $(C^{\mathrm{T}}C*B^{\mathrm{T}}B)$ 和 $C\odot B$，这些中间变量仅使用一次，却需要消耗大量的计算时间与存储空间。若能省去中间变量的计算，减少计算量，就能够提升算法的性能。

3.2 Tensor Khatri-Rao 乘积

本节给出基于 CP-ALS 算法中的模-n 矩阵与 Khatri-Rao 乘积的计算过程，提出简化的 Tensor Khatri-Rao 乘积，并证明简化后的 TKR 乘积与原算法的计算结果相同。

通过分析可知，若能将张量分解过程中多个矩阵间的运算合并，省去中间的变量，则可减少算法的总计算量。从 CP-ALS 算法的复杂度分析中可知，算法的计算量集中在大规模矩阵乘法和 Khatri-Rao 乘积的计算过程。本文将该过程的矩阵运算进行合并和简化，并将其称为 Tensor Khatri-Rao 乘积，简称为 TKR。

首先，将算法 1 的步骤 2 表示为 $A=M_1M_2$，其中 $M_1=X_{(1)}(C\odot B)$，$M_2=(C^{\mathrm{T}}C*B^{\mathrm{T}}B)^{\dagger}$，则 Tensor Khatri-Rao 乘积就是计算 $M_1=X_{(1)}(C\odot B)$ 的过程，也就是计算张量的模-n 矩阵 $X_{(1)}$ 与 B,C 的 Khatri-Rao 结果相乘的过程。之后，给出 TKR 的简化方法，省去计算与存储 $C\odot B$ 的过程，直接给出

$X_{(1)}(C\odot B)$的计算结果,减少了计算量。

TKR 乘积的改进如下:假设张量的模-1 矩阵为 $X_{(1)}\in\mathbb{R}^{I\times(JK)}$,因子矩阵为 $B\in\mathbb{R}^{J\times R}$,$C\in\mathbb{R}^{K\times R}$,若 $M_1=X_{(1)}(C\odot B)$,且 $M_1\in\mathbb{R}^{I\times R}$,则有:

$$M(i,r)=\sum_{j=1}^{JK}[X_{(1)}(i,j)\times C(\lceil j/J\rceil,r)\times B(1+(j-1)\%J,r)] \tag{9}$$

证明:假设 $Y=(C\odot B)$,则可得:

$$Y=[C_{(:,1)}\otimes B_{(:,1)}\ \cdots\ C_{(:,R)}\otimes B_{(:,R)}] \tag{10}$$

其中,$Y(i,r)=C(\lceil j/J\rceil,r)\times B(1+(i-1)\%J,r)$;若 $M=X_{(1)}Y$,且 $X_{(1)}\in\mathbb{R}^{I\times JK}$,$Y\in\mathbb{R}^{JK\times R}$,则 $M(i,r)=\sum_{j=1}^{JK}X(i,j)Y(j,r)$,则式(10)可写为:

$$M(i,r)=\sum_{j=1}^{JK}[X_{(1)}(i,j)\times C(\lceil j/J\rceil,r)\times B(1+(j-1)\%J,r)]$$

3.3　稀疏结构 CSR

本节介绍了本文并行算法中采用的 CSR(Compressed Sparse Row)稀疏结构,以及采用该结构的原因。

张量的高维度特性使得存储张量所需的空间呈指数级增长,如 CP-ALS 算法中存储张量展开的模-n 矩阵 $X_{(n)}$ 的空间复杂度约为 $O(n^3)$。然而,现实中的真实张量数据往往具有很高的稀疏性,因此本文在计算张量分解的过程中采用稀疏的数据结构存储数据。

为了使 GPU 并行计算稀疏矩阵的运算获得更高的效率,出现了许多稀疏的矩阵存储结构,如 COO,ELL,CSR 和 HYB 等[12]。在 CP-ALS 算法中,稀疏程度高的矩阵为模-n 矩阵 $X_{(n)}$,其主要运算为 $M_1=X_{(1)}(C\odot B)$,其中 $X_{(n)}$ 在矩阵乘法的左边,因此采用按行压缩的 CSR 结构进行存储能够有效提升矩阵乘法的计算效率[13]。

利用 CSR 形式存储稀疏矩阵的思想是将矩阵按行划分,并按顺序存储矩阵每行中的非零元素。若稀疏矩阵 $A\in\mathbb{R}^{I\times J}$ 包含 n 个非零元素,则 CSR 形式使用非零元素值 $Val\in\mathbb{R}^n$、列号 $Col\in\mathbb{N}_+^n$ 和偏移量 $RowPtr\in\mathbb{N}_+^{J+1}$ 3 个数组表示稀疏矩阵。假设矩阵中第 m 个非零元素为 $x_{ij}=a$,则满足 $Val_m=a$,$Col_m=j$,并且稀疏矩阵第 i 行的非零元素总数为 $N_i=RowPtr_i-RowPtr_{i-1}$,其中 $RowPtr_0=0$。

图 4 给出一个稀疏矩阵的矩阵形式与其对应的 CSR 稀疏形式的示意图。

$$\begin{bmatrix}1&7&0&0\\0&2&8&0\\5&0&3&9\\0&6&0&4\end{bmatrix}$$

[0,2,4,7,9] = *RowPtr*
[0,1,1,2,0,2,3,1,3] = *Col*
[1,7,2,8,5,3,9,6,4] = *Val*

图 4　稀疏矩阵的 CSR 图例

Fig. 4　CSR example of sparse matrix

在 CP-ALS 算法中,若将张量的模-n 矩阵 $X_{(n)}$ 以稀疏形式存储,则算法的时间复杂度和空间复杂度会相应地减小。已知张量 $\mathcal{Y}\in\mathbb{R}^{n\times n\times n}$ 中非零元素的个数为 m,则计算稀疏矩阵 $X_{(1)}$ 与$(C\odot B)$乘积的时间复杂度约为 $O(2mR)$[11]。若采用稀疏 CSR 形式存储数据,存储稀疏的模-n 矩阵 $X_{(n)}$ 的空间复杂度约为 $O(3m)$,则 CP-ALS 算法的时间复杂度约为 $O(mR+n^2R)$,空间复杂度约为 $O(m+n^2R)$。

之后,将 TKR 乘积应用到采用稀疏存储的 CP-ALS 算法中,称为 Simplified CP-ALS 算法,简称为 SCP-ALS 算法。若张量的模-1 矩阵 $X_{(1)}$ 已经表示为 CSR 稀疏形式,则稀疏矩阵 $X_{(1)}$ 的第 i 行中非零元的个数为 $N_i=RowPtr_i-RowPtr_{i-1}$,进而式(11)可以改为:

$$M_1(i,r)=\sum_{n=0}^{N_i}[Val_{RowPtr_{i-1}+n}\times C(\lceil Col_{RowPtr_{i-1}+n}/J\rceil,r)\times B(1+(Col_{RowPtr_{i-1}+n}-1)\%J,r)]$$

SCP-ALS 算法的复杂度分析如下:由于省去了计算 $C\odot B$ 的中间过程,因此计算 $X_{(1)}$ 与 $C\odot B$ 的 TKR 乘积的时间复杂度约为 $O(4mR)$,则算法的时间复杂度约为$O(mR)$,空间复杂度约为 $O(m+nR)$。

3.4　结合 GPU 技术的并行 CP 张量分解算法

本文在 SCP-ALS 算法的基础上,结合 GPU 的并行特点,先给出线程划分与核函数的实现方式,再给出完整的结合 GPU 的并行 CP 张量分解算法。

因为在 TKR 乘积方法中,计算结果矩阵 M_1 的每个元素的过程中所需的计算量较小且相互独立,为了充分发挥 GPU 的并行计算能力,我们使每一个并行线程按照 TKR 方法计算 M_1 中的一个元素。GPU 内线程划分的方式如下:首先,根据模-1 矩阵 X_1 的行数 I,将 GPU 分为 I 个线程块(Block);然后,根据 M_1 的列数 r,在每个 Block 中划分 r 个线程。对 GPU 线程进行划分,共得到 $I\times r$ 个线程(thread),则 M_1 的每一个元素都有对应的线程为其计算结果。这种线程划分方式能够划分出尽可能多的并行线程,同时使每个并行任务规模的计算量很小。

同时,在 GPU 并行计算过程中会存在若将所有数据全部存入全局内存(global memory)中,则当不同的线程同时调用同一数据时会引起数据使用冲突和线程延迟的问题。为避免这一问题,在上述线程划分后,使用 GPU 为每个 Block 分配的共享内存(shared memory)存储线程块内重复调用的数据[12],也即在第 i 个 Block 的共享内存中存储 $X_{(1)}$ 中第 i 行的非零元素,以避免第 i 个 Block 的 r 个线程在计算中同时调用第 i 个 Block 的非零元素而引发数据调用冲突。

以计算 $M_1=X_{(1)}(C\odot B)$为例,并行的 TKR 乘积的流程图如图 5 所示,其中虚线框表示并行的计算过程。TKR-Kernal 核函数的伪代码如算法 2 所示。

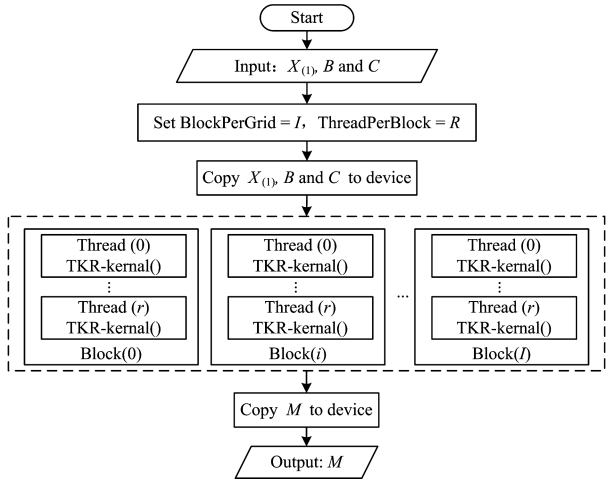


图 5　并行的 TKR 乘积的流程图

Fig 5　Flowchart of parallel TKR product

算法 2 TKR_kernal()

Input: $X_{(1)} \in \mathbb{R}^{I\times(JK)}$, $B\in\mathbb{R}^{J\times R}$, $C\in\mathbb{R}^{K\times R}$, rank R

Output: $M_1 \in \mathbb{R}^{I\times R}$

1. Set bid = blockIdx, tid = threadIdx;
2. Allocate Shared Memory for smCol and smVal of bidth row of $X_{(1)}$;
3. Set N_i as the number of nonzero elements of bidth row of $X_{(1)}$;
4. $N_i = RowPtr_{bid+1} - RowPtr_{bid}$;
5. For n=1:N_i
6. Copy $Col_{RowPtr_{bid}+n}$ to $smCol_n$, $Val_{RowPtr_{bid}+n}$ to $smVal_n$;
7. End For
8. For i=1:N_i
9. $m_{bidtid} += [smVal_i \times c_{\lceil smCol_i/J\rceil tid} \times b_{(1+(smCol_i-1)\%J)tid}]$;
10. End For
11. Free smCol, smVal;
12. Return M_1.

接下来给出完整的结合 GPU 的并行 CP 张量分解算法，称为 Parallel Simplified CP-ALS 算法，简称为 ParSCP-ALS 算法，其伪代码如算法 3 所示。

算法 3 ParSCP-ALS 算法

Input: Tensor $\mathcal{Y}\in\mathbb{R}^{I\times J\times K}$, rank R, Max iteration T, Store the sparse $X_{(n)}$ as CSR format

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

1. For t=1:T
2. $M_1 = TKR(X_{(1)}, B, C)$, $M_2 = (C^TC * B^TB)^\dagger$;
3. $A = M_1 M_2$;
4. Normalize A, λ= norms;
5. $M_1 = TKR(X_{(2)}, C, A)$, $M_2 = (C^TC * A^TA)^\dagger$;
6. $B = M_1 M_2$;
7. Normalize B, λ=norms;
8. $M_1 = TKR(X_{(3)}, B, A)$, $M_2 = (B^TB * A^TA)^\dagger$;
9. $C = M_1 M_2$;
10. Normalize C, λ=norms;
11. If convergence criterion ε is met
12. Break the For loop
13. End If
14. End For
15. Return A, B, C, λ.

4 实验及结果分析

本节介绍实验基于的平台和采用的数据集，给出 ParSCP-ALS 算法的性能对比结果，并进行实验结果分析。

本文中张量分解算法的初始化过程均使用 Matlab 软件的 svds()函数来计算奇异值分解，由于初始化时间相等，因此未将其计入算法的比较时间内。算法中最大的循环次数取为 $T=50$，张量的秩取为 $R=10$，收敛条件取 $\varepsilon=10^{-6}$。

实验所使用的计算机 CPU 处理器为 Intel i7-6700K，4.00GHz，内存为 32GB，GPU 为 NVIDIA GTX1070，显存为 8GB，操作系统为 Win10 64 bit；软件环境为 Visual Studio 2013，CUDA 8.0。

4.1 度量标准

本文使用并行算法相对于串行算法的加速比来评价算法的优劣[12]。定义算法的加速比 s 为：

$$s=\frac{T_s}{T_p}$$

其中，T_s 表示串行算法所需的时间，T_p 表示并行算法所需的时间。

4.2 数据集

为了客观测试并行算法的性能，分别在模拟数据集和真实数据集上进行实验。

由算法复杂度分析可知，ParSCP-ALS 算法的时间复杂度与张量数据的规模和稀疏程度有关，因此我们随机生成了不同规模和不同稀疏程度的张量数据集来进行对比实验。为测试张量 $\mathcal{Y}\in\mathbb{R}^{N\times N\times N}$ 的规模对算法的影响，分别取 N 为 1000，5000 和 10000 的 3 组张量。同时，为了测试稀疏程度对算法时间的影响，每组中分别选取的非零元素个数为 *nonzeros*=$10N$，$50N$，$100N$。

为测试本文的 ParSCP-ALS 算法在真实数据集中的性能，选取 DBLP 引文数据集[14] 和 Movielens 电影评分数据集[15] 进行了算法性能的对比实验。

DBLP 数据集：计算机领域内以作者为核心的英文文献数据集，其中按年代顺序列出了作者的科研成果。本文根据 DBLP 数据集中文献信息中的作者姓名和杂志名称构造作者-作者-杂志的合作关系张量，用于表示不同作者在同一杂志上发表文献的合作情况。本文在 DBLP 数据集 2016 年内所有的文献类型为 article 的文献中，以文献(article)数量为基础随机提取 5 组不同规模的数据，构造不同规模的张量数据。其中，*nonzero* 表示三阶张量中非零元素的数量，也即作者-作者-杂志的合作关系数量。表 1 列出所选取的 5 组 DBLP 数据集的基本信息。

表 1 DBLP 数据集的信息

Table 1 Information of DBLP datasets

DBLP	article	author(I,J)	journal(K)	*nonzero*
D_1	1000	3888	415	6790
D_2	2500	9645	655	18756
D_3	5000	16607	845	35941
D_4	7500	25010	960	49104
D_5	10000	32254	1046	67263

Movielens 电影评分数据：数据中包含不同用户对不同电影的评分和每一部电影的类型标签，评分范围为 1～5。本文通过评分数据构造用户-电影-标签的电影评分的评价信息张量，张量的值为相应评分。数据中含有的电影数量为 3884，用户数量为 6040，电影标签数量为 18，评价总数为 2101816。

4.3 实验结果及分析

在模拟数据集的算法性能对比实验中，本文对 ParSCP-ALS 算法、SCP-ALS 算法、ParCP-ALS 算法和 CP-ALS 算法进行对比，其中 CP-ALS 算法为选取稀疏存储 CSR 表示后的算法，ParCP-ALS 算法为现有的并行张量分解算法，也即在 CP 张量分解的计算过程中直接使用并行的矩阵乘积的算法。本文分别比较了这 4 种张量分解算法在模拟数据集上计算 CP 张量分解过程中执行一次循环所需的运行时间。

表 2－表 4 分别列出 4 种算法在 3 组模拟数据集上的实验结果。

表 2　张量规模为 $N=1000$ 时一次循环所需的时间

Table 2　Running time per cycle with $N=1000$

（单位：ms）

	$10N$	$50N$	$100N$
ParSCP-ALS	22.8	28.2	31.5
SCP-ALS	16.4	31	49.1
ParCP-ALS	47.3	50.7	54
CP-ALS	282	316	347

表 3　张量规模为 $N=5000$ 时一次循环所需的时间

Table 3　Running time per cycle with $N=5000$

（单位：ms）

	$10N$	$50N$	$100N$
ParSCP-ALS	70.8	88.8	98.1
SCP-ALS	75.2	159.4	279.3
ParCP-ALS	844.2	885	901.5
CP-ALS	6610	6836	7106

表 4　张量规模为 $N=10000$ 时一次循环所需的时间

Table 4　Running time per cycle with $N=10000$

（单位：ms）

	$10N$	$50N$	$100N$
ParSCP-ALS	122	149.4	197
SCP-ALS	148	338.6	610.3
ParCP-ALS	2547	2566	2624
CP-ALS	27675	28305	29046

由 3 组实验的结果可得，ParSCP-ALS 算法和 SCP-ALS 算法的计算时间比 ParCP-ALS 算法和 CP-ALS 算法的计算时间都少，其中本文给出的 ParSCP-ALS 算法计算一次循环时的计算效率最高。

数据集 $N=1000$ 的实验中，ParSCP-ALS 算法与 SCP-ALS 算法的运行时间差距较小，当 $nonzeros=10\,N$ 时，ParSCP-ALS 进行一次循环的时间为 22.8 ms，SCP-ALS 进行一次循环的时间仅为 16.4 ms。并行算法的计算时间长于串行算法的计算时间的原因为：ParSCP-ALS 算法的运算时间包括数据在主机（Host）和 GPU（Device）之间的传输时间和并行计算的时间，在张量规模较小的情况下，GPU 内并行计算的线程数量较少，使得数据传输时间长于并行计算相比原算法所节约的时间，从而导致并行算法的总体运算时间较长。随着张量规模的增大与非零元素数量的增加，GPU 并行计算的优势逐步显现，不再出现并行算法比原算法时间长的情况。

表 5 和表 6 分别列出并行 ParSCP-ALS 算法与并行 ParCP-ALS 算法在模拟数据集中的加速比。通过观察表 5 和表 6 可得，ParCP-ALS 算法的加速比总体高于 ParSCP-ALS 算法的加速比，ParCP-ALS 算法在实验中的最低加速比为 5.97，而 ParSCP-ALS 算法的加速比最高为 3.09。虽然 ParSCP-ALS 算法的加速比没有现有并行算法的加速比高，但结合运行时间的实验结果可得，本文算法的计算时间少于现有并行算法，本文的并行算法效率更高。

表 5　并行 ParCP-ALS 算法相对 CP-ALS 的加速比

Table 5　Speedup of parallel ParCP-ALS compared to CP-ALS

（单位：ms）

S	$10N$	$50N$	$100N$
$N=1000$	5.97	6.23	6.46
$N=5000$	7.63	7.72	7.88
$N=10000$	10.86	11.03	11.07

表 6　并行 ParSCP-ALS 算法相对 SCP-ALS 的加速比

Table 6　Speedup of parallel ParSCP-ALS compared to SCP-ALS

（单位：ms）

S	$10N$	$50N$	$100N$
$N=1000$	—	1.09	1.56
$N=5000$	1.06	1.79	2.84
$N=10000$	1.21	2.26	3.09

图 6 显示了 ParSCP-ALS 算法、SCP-ALS 算法与 ParCP-ALS 算法在 DBLP 数据集上的运行时间。CP-ALS 算法在 DBLP 数据集 D_1，D_2 上的运算时间分别为 35 s 和 846 s，与其他算法的时间差距较大，因此未在图中显示。CP-ALS 算法在 DBLP 数据集 D_3，D_4 和 D_5 上运行时，计算所需的内存超过实际内存，未能计算出最终结果。

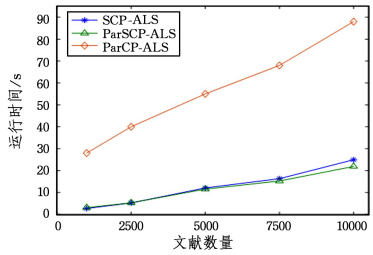


图 6　3 种算法在 DBLP 数据集上的性能比较

Fig. 6　Performance comparison on DBLP datasets of three algorithms

通过图 6 中的实验结果可得，相比于 ParCP-ALS 算法，ParSCP-ALS 算法能够有效地减少计算张量分解的时间。因为由 DBLP 构造的张量数据中非零元素的比例约为 $10^{-6}\sim10^{-8}$，稀疏程度较高，并行计算过程中线程分配和内存调用的时间长于计算所花的时间，导致 ParSCP-ALS 算法与 SCP-ALS 算法的计算时间差距较小。

图 7 显示了 CP-ALS 算法、ParCP-ALS 算法、SCP-ALS 算法和 ParSCP-ALS 算法在 Movielens 数据集上进行 CP 张量分解的运行时间。通过对比可知，ParCP-ALS 算法的加速比约为 2.6，ParSCP-ALS 算法的加速比约为 1.8，其中 ParSCP-ASL 算法相比 ParCP-ALS 算法缩短了约 58% 的计算时间。

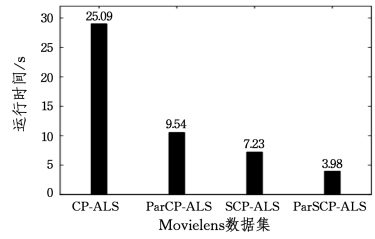


图 7　4 种算法在 Movielens 数据集上的性能比较

Fig. 7　Performance comparison on Movislens datasets of four algorithms

在 DBLP 数据集上的实验说明，本文的 ParSCP-ALS 算法在计算不同规模的张量分解时的性能都优于 ParCP-ALS 算法，并且能够处理大规模的张量分解问题。Movielens 数据集的规模为 $3884\times6040\times18$，与 DBLP 数据集 D_1 的规模近似，其中非零元素所占比例约为 5×10^{-3}，DBLP 中 D_1 的非

（下转第 317 页）

gation:An algorithm for satisfiability[J]. Random Structures and Algorithms,2005,27(2):201-226.

[9] FEIGE U,MOSSEL E,VILENCHIK D. Complete convergence of message passing algorithms for some satisfiability problems [C]// Proceedings of Random06. 2008:339-350.

[10] WANG X F,XU D Y,WEI L. Convergence of Warning Propagation Algorithms for Random Satisfiable Instances[J]. Journal of Software,2013,24(1):1-11. (in Chinese)
王晓峰,许道云,韦立. 随机可满足实例集上警示传播算法的收敛性[J]. 软件学报,2013,24(1):1-11.

[11] WANG X F,XU D Y. Sufficient Conditions for Convergence of the Warning Propagation Algorithm[J]. Journal of Software,2016,27(12):3003-3013. (in Chinese)
王晓峰,许道云. 警示传播算法收敛的充分条件[J]. 软件学报,2016,27(12):3003-3013.

[12] XU D Y. Applications of minimal unsatisfiable formulas to polynomially reduction for formulas[J]. Journal of Software,2006,17(5):1204-1212. (in Chinese)
许道云. 极小不可满足公式在多项式归约中的应用[J]. 软件学报,2006,17(5):1204-1212.

[13] XU D Y,WANG X F. A Regular NP-Complete Problem and Its Inapproximability[J]. Journal of Frontiers of Computer Science & Technology,2013,7(8):691-697. (in Chinese)
许道云,王晓峰. 一个正则 NP-完全问题及其不可近似性[J]. 计算机科学与探索,2013,7(8):691-697.

[14] WANG X F,QIANG L I,DING H S. Convergence of Warning Propagation Algorithm for Regular Structure Instances[J]. Computer Science,2015,42(1):279-284. (in Chinese)
王晓峰,李强,丁红胜. 规则实例集上警示传播算法的收敛性[J]. 计算机科学,2015,42(1):279-284.

(上接第 303 页)

零元素的比例仅约为 10^{-6}。综合两个数据集上的实验结果得出,在面对不同稀疏程度的真实张量数据时,本文给出的 ParSCP-ALS 算法相较于现有并行和串行算法获取得更好的性能。

结束语　本文提出的结合 GPU 技术的并行 CP 张量分解 ParSCP-ALS 算法是对 CP-ALS 张量分解算法进行改进后的并行实现。首先,对 CP-ALS 中的 Tensor Khatri-Rao 乘积进行简化,并给出 SCP-ALS 算法;然后,结合张量数据的稀疏特性与 SCP-ALS 算法,实现原 CP-ALS 算法的并行化,也就是 ParSCP-ALS 算法。基于模拟数据集和真实数据集的算法性能对比实验,都验证了 ParSCP-ALS 算法能有效提升 CP 张量分解的计算效率。模拟数据集和真实数据集上的实验结果都表明,相比于现有的并行张量算法,ParSCP-ALS 算法有着更优的性能,能够高效地处理大规模稀疏张量的分解问题。

参 考 文 献

[1] LI X T. Research on Key Technologies of Multi-Relational Graph Mining Based on Tensors[D]. Harbin:Harbin Institute of Technology,2013. (in Chinese)
李旭涛. 基于张量的多关系图挖掘关键技术研究[D]. 哈尔滨:哈尔滨工业大学,2013.

[2] YANG B,LIU D Y,LIU J M,et al. Complex Network Clustering Algorithms[J]. Journal of Software,2009,20(1):54-66. (in Chinese)
杨博,刘大有,LIU J M,等. 复杂网络聚类方法[J]. 软件学报,2009,20(1):54-66.

[3] MØRUP M. Applications of tensor (multiway array) factorizations and decompositions in data mining[J]. Wiley Interdisciplinary Reviews Data Mining & Knowledge Discovery,2011,1(1):24-40.

[4] HITCHCOCK F L. The expression of a tensor or a polyadic as a sum of products[J]. Journal of Mathematics and Physics,1927,6(1-4):164-189.

[5] KOLDA T G,BADER B W. Tensor Decompositions and Applications[J]. Siam Review,2009,51(3):455-500.

[6] HARSHMAN R A. Foundations of the PARAFAC procedure:Model and conditions for an 'explanatory' multi-mode factor analysis [J]. UCLA Working Papers in Phonetics,1970,16:1-84.

[7] CARROLL J D,PRUZANSKY S,KRUSKAL J B. CANDELINC:a general approach to multi dimension analysis of many-way arrays with linear constraints on parameters[J]. Psychometrika,1980,45(1):3-24.

[8] PAPALEXAKIS E E,FALOUTSOS C,SIDIROPOULOS N D. ParCube:Sparse Parallelizable Tensor Decompositions[J]. Acm Transactions on Knowledge Discovery from Data,2012,10(1):521-536.

[9] ANTIKAINEN J,HAVEL J,JOSTH R,et al. Nonnegative Tensor Factorization Accelerated Using GPGPU[J]. IEEE Transactions on Parallel & Distributed Systems,2011,22(7):1135-1144.

[10] ZOU B,LI C,TAN L,et al. GPUTENSOR:Efficient tensor factorization for context-aware recommendations[J]. Information Sciences,2015,299:159-177.

[11] KANG U,PAPALEXAKIS E,HARPALE A,et al. GigaTensor:scaling tensor analysis up by 100 times-algorithms and discoveries[C]// ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM,2012:316-324.

[12] COOK S. CUDA Programming-A Developer's Guide to Parallel Computing with GPUs[M]. Beijing:China Mechine Press,2014. (in Chinese)
COOK S. CUDA 并行程序设计[M]. 北京:机械工业出版社,2014.

[13] BELL N,GARLAND M. Efficient Sparse Matrix-Vector Multiplication on CUDA[OL]. http://www. users. csbsju. edu/~mheroux/fau2013_csci317/SPMV-Garland-Bell. pdf.

[14] LEY M. DBLP:some lessons learned[J]. Proceedings of the VLDB Endowment,2009,2(2):1493-1500.

[15] HARPER F M,KONSTAN J A. The MovieLens Datasets:History and Context[J]. ACM Transactions on Internactive Intelligent Systmes(TiiS),2015,5(4):19.