基于异构系统的多级并行稀疏张量向量乘算法

陈玥丹[1),4)] 肖国庆[2),3),4)] 阳王东[2),3)] 金纪勇[5)]
龙 军[1)] 李肯立[2),3)]

[1)](中南大学大数据研究院 长沙 410083)
[2)](湖南大学信息科学与工程学院 长沙 410082)
[3)](国家超级计算长沙中心 长沙 410082)
[4)](湖南大学深圳研究院 广东 深圳 518000)
[5)](之江实验室基础理论研究院—应用数学与机器智能研究中心 杭州 311100)

摘 要 张量在许多实际应用中被用来表示大规模、多源、高维、多模态的数据.稀疏张量分解作为挖掘数据中隐藏信息的有效方法之一,已被广泛应用于机器学习、文本分析、生物医疗等研究领域中.稀疏张量向量乘(Sparse Tensor-VectorMultiplication,SpTV)是张量分解中最基础、耗时最多的运算之一.为加速大数据和人工智能相关应用的运行效率,本文提出了基于CPU-GPU异构结构的多级并行SpTV加速算法.首先,为了将SpTV运算映射到混合、多级并行的分布式CPU-GPU异构多/众核构架,本文设计了一种多维并行SpTV划分方法,采用面向节点级并行的N-1维张量划分和面向GPU线程级并行的矩阵划分,充分利用计算节点间和节点内的多级并行计算能力.其次,设计了一种基于稀疏张量纤维的压缩存储格式,压缩稀疏张量的内存占用,优化SpTV运算的计算和访存模式.最后,提出了基于多流并行的异构高效SpTV算法,进一步设计了稀疏张量的细粒度划分方法、多流并行运行机制和基于张量块排序的多流并行优化技术,实现了SpTV运算中通信开销和计算开销的相互重叠与隐藏.实验结果表明,与相关工作aeSpTV相比,所提出的SpTV算法在所有测试数据集上最高能够获得3.28倍的加速比.

关键词 CPU-GPU;异构并行计算;多级并行;稀疏张量;张量运算
中图法分类号 TP301 **DOI号** 10.11897/SP.J.1016.2024.00441

Exploiting Hierarchical Parallelism for Sparse Tensor-Vector Multiplication on Heterogeneous Parallel Systems

CHEN Yue-Dan[1),4)] XIAO Guo-Qing[2),3),4)] YANG Wang-Dong[2),3)] JIN Ji-Yong[5)]
LONG Jun[1)] LI Ken-Li[2),3)]

[1)](*Big Data Institute, Central South University, Changsha* 410083)
[2)](*College of Computer Science and Electronic Engineering, Hunan University, Changsha* 410082)
[3)](*National Supercomputing Center in Changsha, Changsha* 410082)
[4)](*Shenzhen Research Institute, Hunan University, Shenzhen, Guangdong* 518000)
[5)](*Research Center for Applied Mathematics and Machine Intelligence, Zhejiang Lab, Hangzhou* 311100)

Abstract Many application domains give rise to multidimensional data that can be naturally

收稿日期:2022-12-06;在线发布日期:2023-11-04.本课题得到广东省重点领域研究计划(2021B0101190004)、国家自然科学基金(62172157、62202149)、湖南省科技项目(2023GK2002、2021RC3062、2023JJ60002)、广东省自然科学基金(2023A1515012915)、深圳市基础研究面上项目(JCYJ20210324135409026)、之江实验室开放课题(2022RC0AB03)资助.**陈玥丹**(通信作者),博士,副教授,中国计算机学会(CCF)会员,主要研究领域为高性能计算、并行与分布式处理、人工智能与大数据计算.E-mail:chenyuedan@hnu.edu.cn.**肖国庆**(通信作者),博士,教授,中国计算机学会(CCF)高级会员,主要研究领域为高性能计算与智能计算.E-mail:xiaoguoqing@hnu.edu.cn.**阳王东**,博士,教授,长江学者特聘教授,主要研究领域为并行计算.**金纪勇**,硕士,助理研究员,主要研究领域为分布式计算.**龙 军**,博士,教授,中国计算机学会(CCF)杰出会员,主要研究领域为大数据计算与智能软件系统.**李肯立**,博士,教授,国家杰出青年科学基金入选者,中国计算机学会(CCF)会士,主要研究领域为高性能计算系统软件与应用.

represented via tensors. The tensors used in most real-world applications are extremely large and very sparse. The sparse tensor decomposition is an effective approach to predict the unobserved data and is commonly used in machine learning, text analysis, healthcare analytics, and numerous other applications. Sparse tensor-vector multiplication (SpTV) is one of the most fundamental and time-intensive operations in computing tensor decomposition. In order to improve the efficiency of related applications, this paper exploits the hierarchical parallelism for SpTV on CPU-GPU heterogeneous parallel computing systems. First of all, we propose a multidimensional partitioning method to map parallel SpTV to the underlying CPU-GPU heterogeneous and parallel computing architectures. It utilizes the $N-1$-dimensional tensor partitioning to exploit the inter-node parallelism and the matricized tensor partitioning to exploit the intra-node parallelism. Second, based on the multidimensional data partitioning, we design a fiber-wise compressed storage format for sparse tensors to reduce the memory footprint and optimize the computing and memory accessing patterns in parallel SpTV. Third, we design the parallel streaming SpTV algorithm, by adopting the fine-grained data partitioning method, the parallel streaming execution scheme, and the tensor block sorting technique, to overlap the data swapping cost and the computation overhead and further leverage the computing power of GPUs. The experimental results show that the parallel and efficient SpTV algorithm achieves the speedup of up to 3.28 compared to state-of-the-art (aeSpTV) on a CPU-GPU system.

Keywords CPU-GPU; heterogeneous and parallel computing; hierarchical parallelism; sparse tensors; tensor operations

1 引 言

张量是多维度数据表现形式,稀疏张量中的大部分元素为零,因此零元素不需要存储并参与相关计算.稀疏张量通常在人工智能[1-3]、数据挖掘与分析[4-6]、医疗保健[7-9]等很多实际应用中表示大规模、多源、多模态的数据,因此获得了许多研究者的关注.例如,电子邮件的4个属性{主题,发件人,收件人,时间}可以用一个4维张量来表示,其中每封邮件的某个属性值对应于该4维张量中每个非零元素在相应维度上的取值.张量的分解可以挖掘出张量数据中的隐藏信息,因此广泛地应用于推荐系统、会话检测等实际应用.

张量CANDECOMP/PARAFAC(CP)分解[10-12]是应用最广泛的张量分解方法之一.计算CP分解最常用的方法是基于交替最小二乘算法(Alternating Least Aquares, ALS)进行的,该方法简称为CP-ALS. CP-ALS算法采用一系列张量运算,通过每次迭代为张量的每个维度计算一个新的因子矩阵.具体来说,对于一个3阶张量$\mathcal{X}\in\mathrm{R}^{I\times J\times K}$(秩为$R$)的CP分解计算,需要获得3个稠密的因子矩阵$A\in\mathrm{R}^{I\times R}$、$B\in\mathrm{R}^{J\times R}$和$C\in\mathrm{R}^{K\times R}$.张量$\mathcal{X}$的展开矩阵可以由$A$、$B$和$C$表示:

$$\boldsymbol{X}_{(1)}\approx\boldsymbol{A}(\boldsymbol{C}\odot\boldsymbol{B}),\boldsymbol{X}_{(2)}\approx\boldsymbol{B}(\boldsymbol{C}\odot\boldsymbol{A}),\ \boldsymbol{X}_{(3)}\approx\boldsymbol{C}(\boldsymbol{B}\odot\boldsymbol{A}) \tag{1}$$

其中,$\boldsymbol{X}_{(1)}$、$\boldsymbol{X}_{(2)}$和$\boldsymbol{X}_{(3)}$分别为张量$\mathcal{X}$在三个维度上的展开矩阵(如第2.1节介绍),运算符$\odot$被称为Khatri-Rao乘积(Khatri-Rao Product),下式给出了$\boldsymbol{A}\odot\boldsymbol{B}$运算示例:

$$\boldsymbol{A}=\begin{bmatrix}a_{11} & a_{12}\\ a_{21} & a_{22}\end{bmatrix},\boldsymbol{B}=\begin{bmatrix}b_{11} & b_{12}\\ b_{21} & b_{22}\\ b_{31} & b_{32}\end{bmatrix},$$
$$\boldsymbol{A}\odot\boldsymbol{B}=\begin{bmatrix}a_{11}b_{11} & a_{12}b_{12}\\ a_{11}b_{21} & a_{12}b_{22}\\ a_{11}b_{31} & a_{11}b_{32}\\ a_{21}b_{11} & a_{22}b_{12}\\ a_{21}b_{21} & a_{22}b_{22}\\ a_{21}b_{31} & a_{22}b_{32}\end{bmatrix} \tag{2}$$

基于ALS的CP分解通过迭代运算求得$\boldsymbol{A}$、$\boldsymbol{B}$和$\boldsymbol{C}$.例如,在计算$\boldsymbol{A}$时,每次迭代相当于进行一个最小二乘问题的求解,该问题的最优解形式为

$$\hat{A}=\boldsymbol{X}_{(1)}(\boldsymbol{C}\odot\boldsymbol{B})(\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}*\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{\dagger} \tag{3}$$

其中,$(\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}*\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})$是一个$I\times R$的矩阵,运算$(\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}*\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{\dagger}$是求该矩阵的逆.因此,与运算$\boldsymbol{X}_{(1)}(\boldsymbol{C}\odot\boldsymbol{B})$(被称为MTTKRP (Matricized Tensor Times Khatri-Rao Product))相比,运算$(\boldsymbol{C}^{\mathrm{T}}\boldsymbol{C}*\boldsymbol{B}^{\mathrm{T}}\boldsymbol{B})^{\dagger}$涉及了很少的

计算量．可以看出，MTTKRP运算 $\boldsymbol{X}_{(1)}(\boldsymbol{C}\odot\boldsymbol{B})$ 是CP分解中的主要性能瓶颈．同时，MTTKRP的本质就是进行 I 次SpTV运算，因此实际应用中张量CP分解的运算效率取决于SpTV的性能．由于SpTV运算中较低的计算/访存比和不规则的数据索引，已成为计算张量CP分解所面临的最基础的性能挑战．

在现实应用中，稀疏张量的规模非常大．为了支持大规模实际应用中稀疏张量数据的存储与计算，越来越多的工作利用分布式高性能计算系统，将大规模张量划分为较小的数据块，并分布到多个计算节点上处理．目前主流的高性能计算系统设计趋势是采用"主处理器＋加速器"的异构多/众核计算构架（例如CPU-GPU），这种高性能计算系统的大部分计算能力都来自加速器，因此，在设计高效、可扩展算法时，充分挖掘加速器中强大的计算性能，以实现目标计算任务的效率是至关重要的．

大规模SpTV运算在分布式异构多/众核系统上的并行加速主要面临着3个主要的挑战：

（1）如何有效地将SpTV运算映射到目标分布式异构多/众核构架．

分布式异构多/众核系统提供了混合、多级的并行计算能力，包括计算节点间的并行性以及节点内主处理器与加速器提供的混合并行性．数据划分是并行化SpTV运算、开发平台构架混合、多级并行性的重要方法之一．

许多研究提出了面向分布式异构多/众核构架的稀疏张量划分模型，在降低稀疏张量的通信成本的同时，保证其在所有计算节点上的计算成本．Karsavuran等人[13]提出一种针对稀疏张量数据的中粒度划分的超图模型，该模型将张量划分为非零元素不相交的子张量，然后对每个子张量构造一个模相关的粗粒超图，从而将通信量最小化．aeSpTV[14]采用一种基于目标平台的计算构架和内存结构自适应的稀疏张量划分方法，缓解SpTV运算中不规则访存的高开销、并行写冲突、额外的计算量和中间结果爆炸等问题．ALTO[15]将张量的非零元素划分到不同分块中，这些分块的空间坐标可能存在重叠，为了解决这些重叠可能产生的更新冲突，ALTO在多维空间中定位它们的位置，并根据张量数据的重用性自动选择适当的同步机制．CASpMV[16]采用了一种基于描述稀疏矩阵结构特征统计模型的自动调节的稀疏矩阵划分方法，通过分析非零元素分布特征和目标计算平台的构架特征，将稀疏矩阵均匀地分布到不同的计算节点上．

然而，现有针对SpTV加速的稀疏张量划分模型没有考虑通过细粒度张量数据划分来发挥软件流水并行技术，隐藏对细粒度数据块的数据传输时间和并行计算时间，进一步利用目标平台的计算能力．

（2）如何更高效地压缩并存储大规模稀疏张量．

实际应用中的张量数据具有非常稀疏、非零元素分布不规则的特点．一方面，在基于异构并行计算系统的SpTV运算中，稀疏张量的存储方式决定了内存占用量，因此影响了计算节点间数据传输和节点内主处理器与加速器之间数据交换的通信效率．另一方面，适合的稀疏张量存储结构能够优化SpTV运算中的不规则计算和访存模式，从而提高并行SpTV算法的总体运算效率．

很多相关学者提出了稀疏张量存储格式，以及基于这种存储格式实现的稀疏张量运算算法．Qin等人[17]设计了能够支持多种稀疏张量存储格式（包括COO（Coordinate），CSR（Compressed Sparse Row），CSC（Compressed Sparse Column）[18]，DIA（Diagonal）[19]，ZVC（Zero-Value Compression）[20]），HICOO（Hierarchical Coordinate）[21]，CSF（Compressed Sparse Fiber）[22-23]的加速器硬件扩展．

其中，ZVC[20]为每 N 个非零元素生成一个 N 位掩码，其中掩码为"0"表示对应的张量元素值为零，而"1"表示该元素为非零值．生成 N 位掩码后，再将追加存储张量非零元素．因此，N 个连续的张量零元素可以压缩为一个 N 位全零掩码，进而获得 N 倍的压缩比．然而，ZVC只考虑了张量零元素的压缩存储，没有考虑非零元素和张量分块相关索引数据的压缩存储．HICOO[21]对稀疏张量分块中索引数据进行压缩，并使用更短的整数类型来表示张量分块的偏移量．但HICOO格式没有进一步对张量纤维维度的索引进行压缩．CSF[22-23]根据稀疏张量运算基于纤维的计算特性，利用压缩、分层和基于纤维的树形结构表示稀疏张量．但是基于CSF数据结构的张量运算总计算量会随着纤维片段的增加而增加[24]．Dun等人[25]基于分块和位映射设计了一种新的稀疏矩阵压缩存储格式TB-COO，TB-COO压缩了张量切面维度的索引，但在还原压缩索引时，需要进行一系列位操作．Sparta[26]使用多维、高效的哈希表表示SpTC运算的输入稀疏张量．然而，基于哈希表的稀疏张量表示增加了运算中的索引开销，在不擅长访存密集型任务和逻辑运算的加速器（如

GPU等)上,很难发挥良好的运算加速效果. Qiu等人[27]假设给定的大张量具有低多线性秩,提出使用Tucker模型对原始大张量进行压缩表示,从而有效地执行张量分解.

(3)基于给定的任务划分方法和稀疏张量压缩存储格式,如何充分利用高性能异构并行计算系统的计算能力,优化SpTV运算性能.

因此,需要基于并行任务划分和数据存储结构的设计,考虑计算平台上的并行计算效率与节点间、节点内的通信效率,对SpTV算法进行并行化设计与优化,充分发挥目标平台的计算能力.

为应对和缓解上述挑战,本文基于分布式异构CPU-GPU计算系统,设计了多级并行的SpTV算法. 本文主要包括四个贡献:

(1)针对上述第1个挑战,本文设计了一种多维并行SpTV划分方法. 面向分布式异构CPU-GPU计算系统,采用面向节点级并行的N-1维张量划分和面向GPU线程级并行的矩阵划分,降低张量的稀疏度的同时,将计算任务更均匀地映射到多级并行计算架构上,避免了节点间通信和额外的累加,减少了节点内的通信开销.

(2)针对上述第2个挑战,本文为稀疏张量设计了一种基于张量纤维的压缩存储格式,根据SpTV运算属性及设计的多维并行SpTV划分方法,按稀疏张量纤维对非零元素进行压缩存储,进而减少稀疏张量的内存占用,提高了多维并行SpTV运算的数据索引效率.

(3)针对上述第3个挑战,本文面向分布式异构CPU-GPU计算系统,设计了基于张量块排序的多流并行优化策略. 首先,为了充分利用多CUDA流的并行性,本文基于多维并行SpTV划分方法,进一步设计了对稀疏张量的细粒度划分;并设计了多流并行运行机制,将对基于细粒度数据块的并行SpTV运算所产生的通信开销与计算开销进行相互重叠与隐藏;最后,提出了基于张量块排序策略,进一步提高张量块处理开销之间的重叠程度.

(4)本文选取了5个实际应用中的真实稀疏张量数据集,在CPU-GPU构架上测试了多级并行SpTV算法的性能. 与现有相关aeSpTV工作相比,多级并行SpTV算法的吞吐量(每秒10亿次的浮点运算数(Giga Floating-point Operations Per Second, GFLOPS))最高获得了3.28倍的加速比.

2 研究背景

2.1 张量及其相关基本操作

张量的阶(Order)或模(Mode)表示其维度的数量. 向量属于阶为1的张量,矩阵属于阶为2的张量. 阶数不小于3的张量被称为高阶张量.

一个N阶张量的纤维(Fiber)通过固定$N-1$个维度上的索引值得到. 图2分别展示了图1中的3阶张量$\mathcal{X}\in \mathrm{R}^{I\times J\times K}$在3个维度上的纤维. 其中,模1纤维表示为$\boldsymbol{X}_{:jk}$,模2纤维表示为$\boldsymbol{X}_{i:k}$,模3纤维表示为$\boldsymbol{X}_{ij:}$,其中,$i\in\{1, 2, \cdots, I\}$,$j\in\{1, 2, \cdots, J\}$,$k\in\{1, 2, \cdots, K\}$.

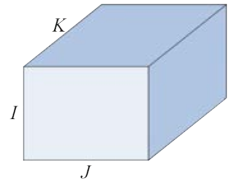

图1 3阶张量$\mathcal{X}\in \mathrm{R}^{I\times J\times K}$

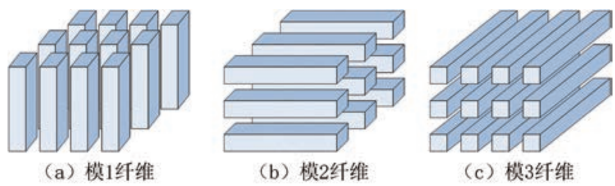

图2 张量纤维

一个N阶张量的切片(Slice)通过固定$N-2$个维度上的索引获得. 图3分别展示了图1中的3阶张量$\mathcal{X}\in \mathrm{R}^{I\times J\times K}$在3个维度上的切片. 其中,第$i$个模1切片表示为$\boldsymbol{X}_{i::}$,第$j$个模2切片表示为$\boldsymbol{X}_{:j:}$,第$k$个模3切片表示为$\boldsymbol{X}_{::k}$,其中,$i\in\{1, 2, \cdots, I\}$,$j\in\{1, 2, \cdots, J\}$,$k\in\{1, 2, \cdots, K\}$.

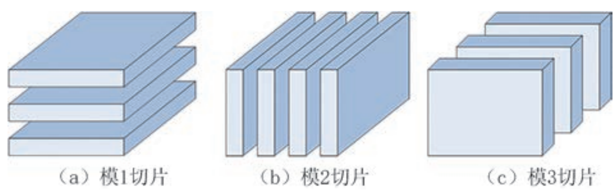

图3 张量切片

张量的矩阵化(Matricization)是将张量展开为一个2阶矩阵的操作. 一个N阶张量的模n矩阵化操作以模n纤维作为矩阵的列将张量展开,展开的张量记为$\boldsymbol{X}_{(n)}$,张量中的元素$(i_1, i_2, \cdots, i_N)$对应于模n展开矩阵中的元素(i_n, j),其中

$$j=\sum_{k=1,k\neq n}^{N} i_k \prod_{m=1,m\neq n}^{k-1} I_m \qquad (4)$$

以一个2×3×2的3阶张量$\mathcal{X}$为例．$\mathcal{X}$的模3切片分别为

$$\boldsymbol{X}_{::0}=\begin{bmatrix}1 & 3 & 5\\ 2 & 4 & 6\end{bmatrix},\boldsymbol{X}_{::1}=\begin{bmatrix}7 & 9 & 11\\ 8 & 10 & 12\end{bmatrix} \qquad (5)$$

那么，$\mathcal{X}$的模1、模2和模3展开矩阵分别为

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

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

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

一个N阶张量$\mathcal{X}\in\mathrm{R}^{I_1\times I_2\times\cdots\times I_N}$与一个长度为$I_n$的向量$\boldsymbol{v}$的模$n$乘(Mode-$n$ Multiplication)记为$\mathcal{X}\times_n\boldsymbol{v}$，计算结果是一个$N-1$阶张量$\mathcal{Y}\in\mathrm{R}^{I_1\times I_2\times\cdots\times I_{n-1}\times I_{n+1}\times\cdots\times I_N}$，即对$\mathcal{X}$的每个模$n$纤维与$\boldsymbol{v}$执行内积运算而获得．例如，式(5)中表示的3阶张量$\mathcal{X}$与向量$\boldsymbol{v}=\{1,2\}$的模1乘积为

$$\mathcal{X}\times_1 v=\begin{bmatrix}5 & 23\\ 11 & 27\\ 17 & 30\end{bmatrix} \qquad (9)$$

2.2 异构编程模型

随着机器学习和数据挖掘等实际应用中计算负载的剧增，越来越多的计算系统配备了GPU加速器．GPU通常包含了超过2000个计算核(线程)，远远超过了现有的多核CPU处理器．在典型的CPU-GPU异构构架中，CPU和GPU都配备了各自的私有内存，分别称为主机内存(Host Memory)和设备内存(Device Memory)，并通过PCIe(Peripheral Component Interconnect-Express)互联互通．

统一计算设备架构CUDA(Compute Unified Device Architecture)为开发人员提供了利用NVIDA GPU计算资源的并行编程框架．CUDA程序会将运行的kernel函数中设置的线程块调度到GPU的流多处理器(Stream Multiprocessor，SM)上并发进行．CUDA处理流程主要包括4个步骤：

(1)将数据从主机内存复制到设备内存；

(2)CPU启动GPU上的kernel函数；

(3)GPU的CUDA线程并行处理kernel函数；

(4)将设备内存上的计算结果传回主机内存．

本文针对一个N阶稀疏张量$\mathcal{X}\in\mathrm{R}^{I_1\times I_2\times\cdots\times I_N}$，介绍其SpTV模$n$运算在大规模异构众核CPU-GPU集群上的高效算法，其中$n\in\{1,2,\cdots,N\}$．

3 多维并行SpTV划分方法

大规模异构众核CPU-GPU计算系统提供了多级并行计算能力：大量的CPU-GPU节点提供了节点级并行计算能力，每个节点中的GPU能够提供线程级并行计算能力．为了充分利用计算系统的多级并行性，本文对稀疏张量$\mathcal{X}$的SpTV模n运算提出了一种多维并行划分方法．

多维并行SpTV划分方法主要包括面向节点级并行的$N-1$维张量划分和面向节点内GPU线程级并行的模n展开矩阵划分两个主要的部分．

3.1 面向节点级并行的$N-1$维张量划分

如第2.1节所述，SpTV模n运算是依次对输入张量$\mathcal{X}$的每个模n纤维与输入向量$\boldsymbol{v}$执行内积运算．在对运算进行面向α个计算节点的并行数据划分时，如果在模n切片的维度将$\mathcal{X}$划分为α个子张量(其中第i个子张量块记为$sub\mathcal{X}_i\in\mathrm{R}^{I_1,I_2,\cdots,I_{n-1},I_n^{(i)},I_{n+1},\cdots,I_N}$ ($i=\{1,2,\cdots,\alpha\}$)，并且$I_n=\sum_{i=1}^{\alpha}I_n^{(i)}$，那么每个节点上的计算结果需进一步传输到主节点进行最后的累加操作，这样的划分方法造成了节点间通信和额外的计算操作．为了解决这个问题，面向节点级并行的$N-1$维张量划分方法分别在张量模m切片的维度对$\mathcal{X}$进行$N-1$个维度的划分，其中$m=\{1,2,\cdots,n-1,n+1,\cdots,N\}$，那么每个节点上获得的是结果张量的一个相应分块，避免了节点间通信和额外的累加计算．

具体而言，面向节点级并行的$N-1$维张量划分方法根据$\mathcal{X}$各个维度的大小和计算节点的数量(记为α)，将N阶稀疏张量$\mathcal{X}$划分为α个N阶子张量块，其中第i个子张量块记为$sub\mathcal{X}_i\in\mathrm{R}^{I_1^{(i)},I_2^{(i)},\cdots,I_{n-1}^{(i)},I_n,I_{n+1}^{(i)},\cdots,I_N^{(i)}}$ ($i=\{1,2,\cdots,\alpha\}$)，并且$I_1^{(1)}=I_1^{(2)}=\cdots=I_1^{(\alpha)}$，$I_2^{(1)}=I_2^{(2)}=\cdots=I_2^{(\alpha)}$，…，$I_N^{(1)}=I_N^{(2)}=\cdots=I_N^{(\alpha)}$．每个计算节点中的CPU读取一个子张量块$sub\mathcal{X}_i$至主存，该节点负责对$sub\mathcal{X}_i$与输入向量$\boldsymbol{v}$进行SpTV模$n$乘运算，结果为一个$N-1$维张量块$sub\mathcal{Y}_i\in\mathrm{R}^{I_1^{(i)}\times I_2^{(i)},\cdots,I_{n-1}^{(i)},I_{n+1}^{(i)},\cdots,I_N^{(i)}}$．

图4以一个3阶张量$\mathcal{X}\in\mathrm{R}^{I\times J\times K}$与向量$\boldsymbol{v}\in\mathrm{R}^{I}$的SpTV模1乘为例，其中计算节点数量设为$\alpha=4$．如图所示，分别在模2切片和模3切片的维度将$\mathcal{X}$划分为4个子张量$sub\mathcal{X}_1\in\mathrm{R}^{I\times J^{(1)}\times K^{(1)}}$，$sub\mathcal{X}_2\in\mathrm{R}^{I\times J^{(2)}\times K^{(2)}}$，$sub\mathcal{X}_3\in\mathrm{R}^{I\times J^{(3)}\times K^{(3)}}$和$sub\mathcal{X}_4\in\mathrm{R}^{I\times J^{(4)}\times K^{(4)}}$，其中$J^{(1)}=$

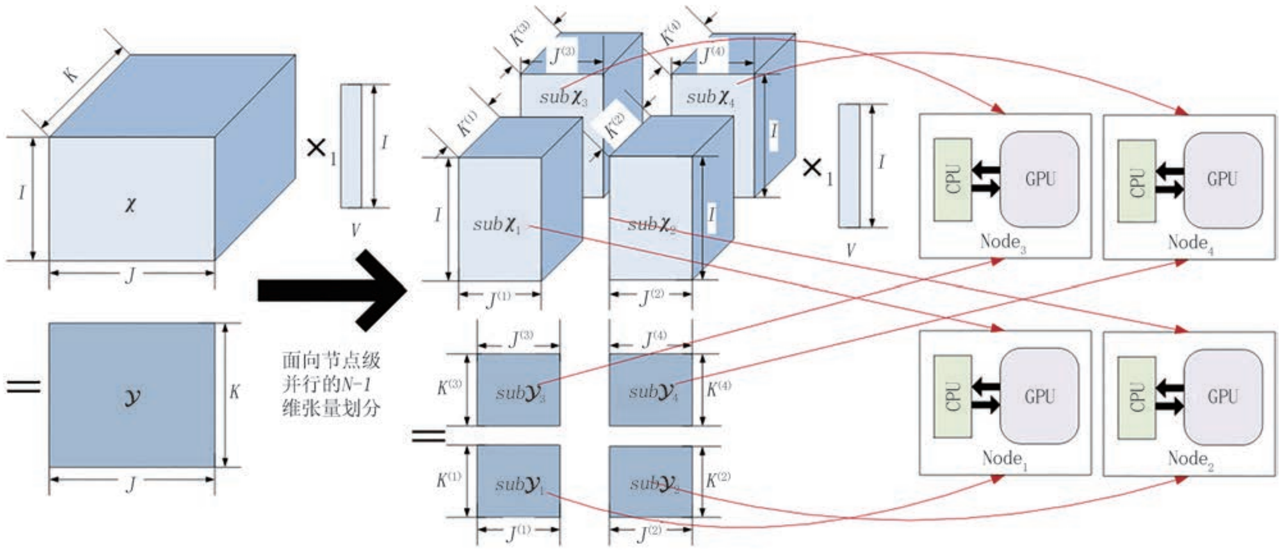

图4 面向节点级并行的$N-1$维张量划分，其中假设计算节点数量设为$\alpha=4$

$J^{(2)}=J^{(3)}=J^{(4)}, K^{(1)}=K^{(2)}=K^{(3)}=K^{(4)}$. 每个计算节点对一个子张量块和向量$\boldsymbol{v}$执行SpTV运算.

3.2 面向GPU线程级张量划分

如第2.2节所述，为了利用GPU资源，首先需要将输入张量和向量数据从CPU主机内存传输到GPU设备内存，接着将计算任务分配给GPU线程，待GPU上的计算完成后，计算结果需从设备内存传回主机内存．此外，根据SpTV的模n乘运算属性，张量中每个模n纤维与向量$\boldsymbol{v}$的内积结果对应的是结果张量$\mathcal{X}$中的一个元素．但稀疏张量中经常存在空模n纤维，即所有元素都为零的模n纤维，在运算中这些空纤维对应于结果张量$\mathcal{Y}$中的元素也为零．为了减少主机内存与设备内存之间的数据传输量、更均匀地为GPU划分计算任务，本文提出了面向GPU线程级并行的矩阵划分方法，通过删除空纤维、降低张量的稀疏度，减少主机内存与设备内存之间的数据传输量，同时更均匀地划分输入张量．

本文设计面向GPU线程级并行的矩阵划分方法，为每个计算节点内的GPU线程划分并行任务．该划分方法主要包括3个步骤：

(1)将分配给每个计算节点的张量数据块$sub\mathcal{X}_i$展开为模n展开矩阵；

(2)将该模n展开矩阵中所有元素都为零的模n纤维(称为空纤维)删除，并记该矩阵为$\boldsymbol{X}_i \in I_n \times M_i$；

(3)根据$\boldsymbol{X}_i$中矩阵列的数量(M_i)和GPU线程的数量(记为β)，继续将$\boldsymbol{X}_i$在矩阵列的维度进行均匀地划分，得到β个矩阵块，每个矩阵块中包含了$M_i/(\alpha\times\beta)$列．本文记$\boldsymbol{X}_i$中的第j个矩阵块为$\boldsymbol{X}_i^{(j)} \in I_n \times M_i/(\alpha\times\beta)$，其中$j\in\{1, 2, \cdots, \beta\}$．第$i$个计算节点中每个GPU线程对$\boldsymbol{X}_i^{(j)}$与输入向量$\boldsymbol{v}$进行SpTV模$n$乘运算．

图5展示了图4所示第1个计算节点内面向GPU线程级并行的矩阵划分示例，其中该节点内的GPU线程数设为$\beta=2$．首先$sub\mathcal{X}_1$展开为模1展开

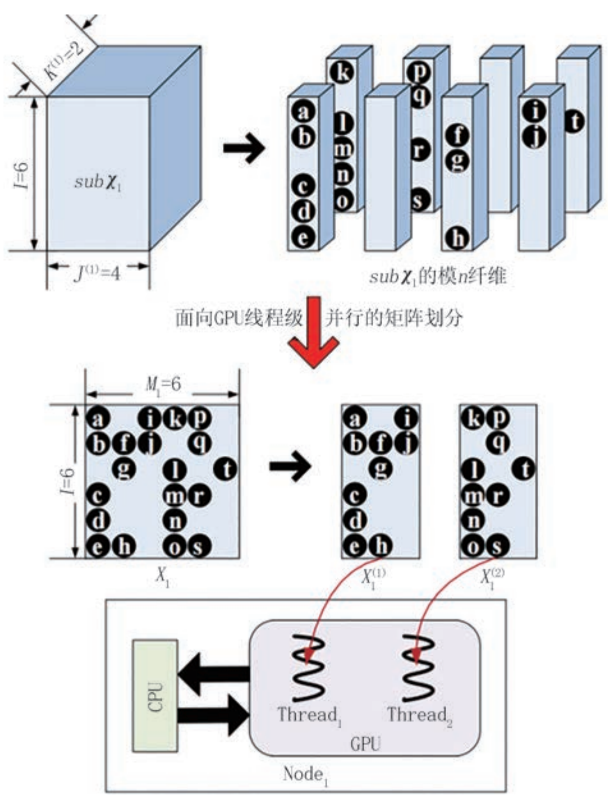

图5 图4所示第1个计算节点内面向GPU线程级并行的矩阵划分示例，其中假设该节点内的GPU线程数设为$\beta=2$

矩阵 $\boldsymbol{X}_1 \in \mathrm{R}^{I\times M_1}$，其中 $I=6$ 并且 $M_1=6$；接着对 $\boldsymbol{X}_1$ 按列划分为 $\beta=2$ 个矩阵块 $\boldsymbol{X}_1^{(j)}$，每个 $\boldsymbol{X}_1^{(j)}$ 中包含了3列．该计算节点内的每个GPU线程对一个矩阵块和向量 $\boldsymbol{v}$ 进行SpTV运算．

4　稀疏张量纤维压缩存储格式

稀疏张量的存储取决于数据的划分策略和计算模式，适配于划分策略的张量存储结构能够提高计算节点和线程对数据块的索引效率．根据本文提出的多维并行SpTV划分方法，各计算节点和GPU线程都是按模 n 纤维对张量数据进行访问．因此，本文采用一种稀疏张量纤维压缩存储格式，按模 n 纤维对张量中的非零元素进行存储，从而适配设计的多维并行SpTV划分方法划分策略，提高数据索引效率以及减少内存占用．

稀疏张量纤维压缩存储格式采用4个数组对每个计算节点上的子稀疏张量块 $sub\mathcal{X}_i$ 进行存储：

（1）存储 $sub\mathcal{X}_i$ 中每个非零模 n 纤维中非零元素数量的数组 $\boldsymbol{P}_f$；

（2）存储每个非零元素在相应模 n 纤维中的索引的数组 $\boldsymbol{Ind}$；

（3）存储每个非零元素数值的数组 $\boldsymbol{Val}$；

（4）存储 $sub\mathcal{X}_i$ 的每个模 m 切片中非零模 n 纤维数量的数组 $\boldsymbol{N}_f$（$m\in\{1,2,\cdots,n-1,n+1,\cdots,N\}$）．

以图5中的子张量块 $sub\mathcal{X}_1$ 为例，稀疏张量纤维压缩存储格式使用4个数组存储 $sub\mathcal{X}_1$：

（1）$\boldsymbol{P}_f[10]=\{0, 5, 5, 8, 10, 15, 19, 19, 20\}$ 存储非零元素的数量，其中 $\boldsymbol{P}_f[r]$-$\boldsymbol{P}_f[r\text{-}1]$ 表示第 r 个非零模1纤维中非零元素的数量；

（2）$\boldsymbol{Ind}[20]=\{1, 2, 4, 5, 6, 2, 3, 6, 1, 2, 1, 3, 4, 5, 6, 1, 2, 4, 6, 3\}$ 存储20个非零元素在模1纤维中的索引；

（3）$\boldsymbol{Val}[20]=\{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t\}$ 存储20个非零元素的数值；

（4）$\boldsymbol{N}_f[3]=\{0, 3, 6\}$ 存储 $sub\mathcal{X}_1$ 的每个模3切片中非零模1纤维的数量，其中 $\boldsymbol{N}_f[s]$-$\boldsymbol{N}_f[s\text{-}1]$ 表示第 s 个模3切片中非零模1纤维的数量．

基于张量纤维的多维并行SpTV算法如算法1所述，CPU首先为输入稀疏张量 $\mathcal{X}$、输入向量 $\boldsymbol{v}$ 和输出张量 $\mathcal{Y}$ 在主机内存中开辟存储空间（第1-6行），再读取输入数据 $\mathcal{X}$ 和 $\boldsymbol{v}$（第7行），并采用稀疏张量纤维压缩存储格式对稀疏张量 $\mathcal{X}$ 进行存储（第8行）．接着为 $\mathcal{X}$、$\boldsymbol{v}$ 和 $\mathcal{Y}$ 在GPU设备内存中开辟存储空间（第9-13行），再将存储稀疏张量 $\mathcal{X}$ 的3个数组 $\boldsymbol{P}_f$、$\boldsymbol{Ind}$ 和 $\boldsymbol{Val}$ 以及存储输入向量的数组 $\boldsymbol{v}$ 从主机内存传到设备内存（第14-17行）．GPU获得数据后，开始执行并行SpTV运算，计算结果存储到设备内存中的数组 $\boldsymbol{Y\_Device}$ 中（第18行）．并行计算结束后，计算结果再从设备内存传输至主机内存中（第19行）．

算法1. 基于张量纤维的多维并行SpTV算法．

输入：输入稀疏张量 $\mathcal{X}$，输入向量 $\boldsymbol{v}$，GPU并行线程数 β

输出：输出张量 $\mathcal{Y}$

1. *cudaHostAlloc*（$\boldsymbol{P}_f$, ⋯）；//在CPU主机内存中分配 $\mathcal{X}$ 的存储空间；
2. *cudaHostAlloc*（$\boldsymbol{Ind}$, ⋯）；
3. *cudaHostAlloc*（$\boldsymbol{Val}$, ⋯）；
4. *cudaHostAlloc*（$\boldsymbol{N}_f$, ⋯）；
5. *cudaHostAlloc*（$\boldsymbol{v}$, ⋯）；//在CPU主机内存中分配 $\boldsymbol{v}$ 的存储空间；
   cudaHostAlloc（$\boldsymbol{Y}$, ⋯）；//在CPU主机内存中分配 $\mathcal{Y}$ 的存储空间；
6. 读取输入数据 $\mathcal{X}$ 和 $\boldsymbol{v}$；
7. 以稀疏张量纤维压缩存储格式将稀疏张量 $\mathcal{X}$ 存储到主机内存中；
8. cudaMalloc（$\boldsymbol{P}_f\boldsymbol{\_Dev}$, ⋯）；//在GPU设备内存中分配 $\mathcal{X}$ 的存储空间；
9. *cudaHostAlloc*（$\boldsymbol{Ind\_Dev}$, ⋯）；
10. *cudaHostAlloc*（$\boldsymbol{Val\_Dev}$, ⋯）；
11. *cudaHostAlloc*（$\boldsymbol{v\_Dev}$, ⋯）；//在GPU设备内存中分配 $\boldsymbol{v}$ 的存储空间；
    cudaHostAlloc（$\boldsymbol{Y\_Dev}$, ⋯）；//在GPU设备内存中分配 $\mathcal{Y}$ 的存储空间；
12. *cudaMemcpy*（$\boldsymbol{P}_f$, $\boldsymbol{P}_f\boldsymbol{\_Dev}$, ⋯, *HostToDevice*）；//将并行计算所需数据从主机内存传输到设备内存中；
13. *cudaMemcpy*（$\boldsymbol{Ind}$, $\boldsymbol{Ind\_Dev}$, ⋯, *HostToDevice*）；
14. *cudaMemcpy*（$\boldsymbol{Val}$, $\boldsymbol{Val\_Dev}$, ⋯, *HostToDevice*）；
15. *cudaMemcpy*（$\boldsymbol{v}$, $\boldsymbol{v\_Dev}$, ⋯, *HostToDevice*）；
16. *SpTV*（β, $\boldsymbol{P}_f\boldsymbol{\_Dev}$, $\boldsymbol{Ind\_Dev}$, $\boldsymbol{Val\_Dev}$, $\boldsymbol{v\_Dev}$, $\boldsymbol{Y\_Dev}$, ⋯）；//GPU上的并行SpTV；
17. *cudaMemcpy*（$\boldsymbol{Y\_Dev}$, $\boldsymbol{Y}$, ⋯, *DeviceToHost*）；//将计算结果从设备内存传输到主机内存．

5　基于张量块排序的多流并行优化

根据算法1所述，CUDA处理流程主要包括数据传输和并行计算两个主要部分，其中数据传输部分包括从主存加载数据到设备内存以及从设备内存返回结果数据到主存，并行计算部分为GPU上进行

的计算．对于CPU-GPU构架，CPU主机端发出CUDA操作的命令后，不会等待该命令执行完毕，而是立刻执行后续命令，这个特征支持了核kernel函数中数据传输和并行计算两个主要部分之前的并行化．为了利用这种软件可并行性，本文基于细粒度数据划分，采用多流并行运行机制，在每个CPU-GPU计算节点上用多个CUDA流处理细粒度数据块，在并行SpTV中的数据传输部分和并行计算部分之间产生并行性，从而使通信时间与计算时间相互隐藏，减少并行SpTV的运行时间．

同时，由于SpTV中张量数据的稀疏性，各细粒度数据块的内存占用量不均匀，因此，CUDA流每次对各数据块的处理时间存在差异，从而也导致CUDA流之间通信时间与计算时间相互隐藏的效果不佳．为了缓解这个问题，本文进一步提出了基于张量块排序的多流并行优化技术，通过调整各数据块的处理顺序，提高CUDA流之间通信与计算之间的重叠程度，获得更好的多流并行优化效果．

5.1 细粒度数据划分

为了保证每个CUDA流都被分配了并行计算任务，提高多流处理效率，多流并行运行机制需要对张量数据进行更细粒度的划分．细粒度数据划分进行在面向GPU线程级并行划分的步骤1之后，将模n展开矩阵$\boldsymbol{X}_i$在矩阵列的维度按非零元素的个数划分为P个子矩阵块，记为$sub\boldsymbol{X}_i^{(p)}$，每个$sub\boldsymbol{X}_i^{(p)}$包含$M_i/P$列，其中$p=\{1, 2, \cdots, P\}$．细粒度数据划分完成后，再分别对每个$sub\boldsymbol{X}_i^{(p)}$进行面向GPU线程级并行划分中的步骤2操作，将$sub\boldsymbol{X}_i^{(p)}$进一步划分为$\beta$个细粒度矩阵块$\boldsymbol{X}_i^{(j)}$，每个$\boldsymbol{X}_i^{(j)}$包含$M_i/(P\times\beta)$列．因此，模$n$展开矩阵$\boldsymbol{X}_i$最终被划分为$P\times\beta$个矩阵块$\boldsymbol{X}_i^{(j)}$，其中$j\in\{1, 2, \cdots, P\times\beta\}$，在每个CUDA流的任务中，每个GPU线程对对应的一个矩阵块$\boldsymbol{X}_i^{(j)}$进行SpTV运算．

以图5中所展示的面向GPU线程级并行的张量划分为例，图6展示了第1个计算节点内面向多CUDA流并行的细粒度数据划分，其中假设GPU线程数为$\beta=2$以及CUDA流的数量为2．如图6所示，细粒度数据划分将张量的模n展开矩阵$\boldsymbol{X}_1$划分为$P=2$个子矩阵块($sub\boldsymbol{X}_1^{(1)}$和$sub\boldsymbol{X}_1^{(2)}$)，第一个CUDA流负责对$sub\boldsymbol{X}_1^{(1)}$进行并行处理，第二个CUDA流负责对$sub\boldsymbol{X}_1^{(2)}$进行并行处理．接着，进一步将$sub\boldsymbol{X}_1^{(1)}$划分为$\beta=2$个细粒度矩阵块($\boldsymbol{X}_1^{(1)}$和$\boldsymbol{X}_1^{(2)}$)，将$sub\boldsymbol{X}_1^{(2)}$划分为$\beta=2$个细粒度矩阵块($\boldsymbol{X}_1^{(3)}$和$\boldsymbol{X}_1^{(4)}$)，每个细粒度矩阵块中包含了差不多数量的

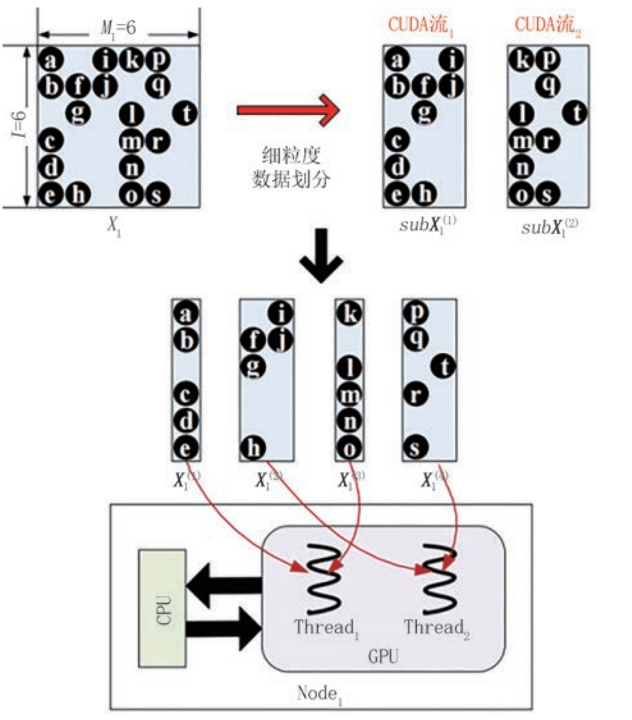

图6 图4所示第1个计算节点内多流并行运行机制中的细粒度数据划分示例(其中假设GPU线程数为$\beta=2$以及CUDA流的数量为2)

矩阵列．因此，在第一个CUDA流的任务中，第一个GPU线程对$\boldsymbol{X}_1^{(1)}$进行计算，第二个GPU线程对$\boldsymbol{X}_1^{(2)}$进行计算；在第二个CUDA流的任务中，第一个GPU线程对$\boldsymbol{X}_1^{(3)}$进行计算，第二个GPU线程对$\boldsymbol{X}_1^{(4)}$进行计算．

5.2 多流并行运行机制

在多流运行机制中，一个CUDA流对一个子矩阵块$sub\boldsymbol{X}_i^{(p)}$进行并行SpTV运算处理的同时，另一个CUDA流从主存加载下一个子矩阵块$sub\boldsymbol{X}_i^{(p+1)}$到GPU设备内存中．因此，多流运行机制在对$sub\boldsymbol{X}_i^{(p)}$进行的并行计算任务与传输$sub\boldsymbol{X}_i^{(p+1)}$的通信任务之间生成了并行性，并行SpTV算法的总体性能获得提高．

每个计算节点使用单个CUDA流时，并行SpTV处理主要包括3个步骤：(1)主存至设备内存：主存中的张量数据$sub\mathcal{X}_i$首先被传输至GPU设备内存，(2)并行计算：接着GPU启用并行线程对$sub\mathcal{X}_i$和向量$\boldsymbol{v}$进行并行SpTV模n乘运算，(3)设备内存至主存：最后将设备内存中的结果张量块$sub\mathcal{Y}_i$传输回主存中．图7显示了第i个计算节点内的GPU多流并行处理机制的示例，其中CUDA流的数量设为2，且在每个CUDA流的每次执行过程中，“主存到设备内存的数据传输”、“并行SpTV运算”和“设备内

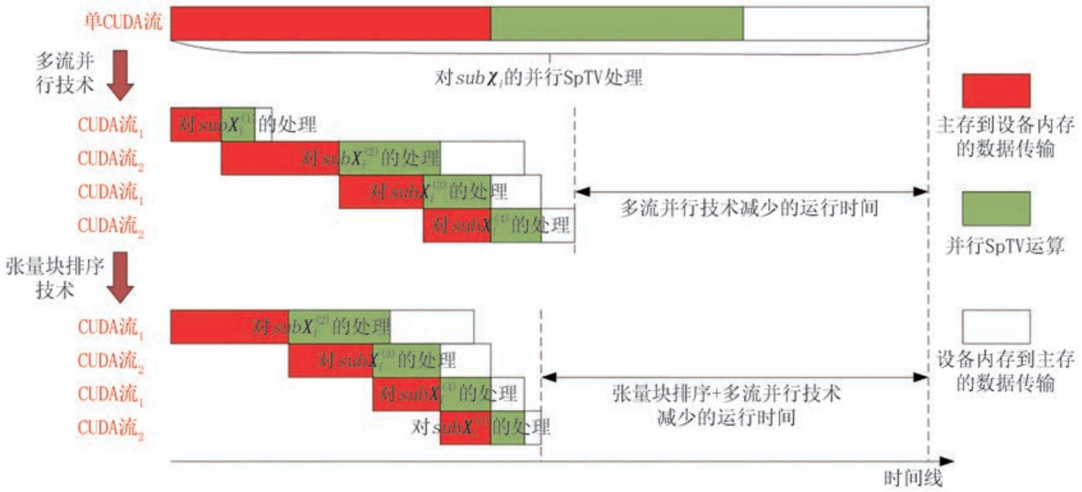

图7　基于多流并行的异构高效优化策略示例(其中假设CUDA流的数量为2)

存到主存的数据传输”三个步骤的耗时并不相等．在使用多流并行处理机制时，张量数据$sub\mathcal{X}_i$被划分为P个子矩阵块$sub\boldsymbol{X}_i^{(p)}$，启用的2个CUDA流轮流对$sub\boldsymbol{X}_i^{(p)}$进行主存至设备内存、并行计算和设备内存至主存的并行SpTV处理．如图7所示，除了将$sub\boldsymbol{X}_i^{(1)}$从主存加载到设备内存的通信无法与并行计算部分相互重叠，对其他子矩阵块的通信都可以与并行计算之间产生并行性，因此减少了并行SpTV的运行时间．

5.3　基于张量块排序的多流并行优化技术

在多流并行运行机制中，各数据块$sub\boldsymbol{X}_i^{(p)}$的处理时间存在差异，因此，对各数据块的处理顺序影响了CUDA流之间通信开销与计算开销的重叠效果．为了获得更好的多流并行优化效果，本文进一步提出了一种基于张量块排序的多流并行优化技术，通过调整各数据块$sub\boldsymbol{X}_i^{(p)}$的处理顺序，提高CUDA流之间通信开销与计算开销的并行度．

如图7中示例所示，4个矩阵块的处理实现存在差异．只采用多流并行技术时，4个矩阵块的处理顺序为：$sub\boldsymbol{X}_i^{(1)}$，$sub\boldsymbol{X}_i^{(2)}$，$sub\boldsymbol{X}_i^{(3)}$，$sub\boldsymbol{X}_i^{(4)}$．张量块排序技术将4个矩阵块的处理顺序按照各数据块的处理时间从大到小进行排序，调整后的处理顺序为：$sub\boldsymbol{X}_i^{(2)}$，$sub\boldsymbol{X}_i^{(3)}$，$sub\boldsymbol{X}_i^{(4)}$，$sub\boldsymbol{X}_i^{(1)}$．如图所示，张量块排序技术能够进一步优化多流并行运行机制的性能．

由于各数据块的处理时间取决于非零元素的数量，因此张量块排序技术按照各数据块中非零元素数量从大到小的顺序对数据块的处理顺序进行排序，提高并行运行机制的优化效果．

6　性能测试与分析

6.1　实验环境

本文基于异构并行计算构架对多级并行的SpTV算法性能进行测试与分析，实验采用的异构并行计算平台中包含了一个14核的E5-2680 CPU@2.40 GHz和一个内存为16 GB的NVIDIA Tesla P100 GPU．所有的实验中，多级并行SpTV算法测试的是张量与向量的模1乘积．

多级并行的SpTV算法在6个多阶的真实张量数据集上进行测试，包括5个3阶张量和1个4阶张量．表1中展示了这6个数据集的信息，其中I_1、I_2、I_3和I_4分别表示数据集在4个维度上的维度大小，NNZ表示数据集中非零元素的个数．数据集ratings-m1和ratings-m20来自电影推荐系统MovieLens中的用户对电影的评分数据，进而形成表示用户-电影-评分的3阶张量[28]．数据集user_ratedmovies结合了MovieLens数据集及其相应的网页Internet Movie Database (IMDb)①和电影评价系统Rotten Tomatoes②中的真实数据，构建了用户-电影-评分的3阶张量．数据集user_taggedartists包含了在线音乐系统Last.fm③中2千个用户对音乐艺术家的定义的标签数据，即一个表示用户-音乐艺术家-标签的3阶张量．数据集Submissions来自Kaggle④社区和活动中收集的丰富数据(包括竞赛、用户、提交分数和内核的公共

① http://www.imdb.com/
② http://www.rottentomatoes.com/
③ http://www.last.fm/
④ https://www.kaggle.com/

数据). 数据集Uber_Pickups来自NYC Taxi & Limousine Commission (TLC)[⑤],是一个4阶张量,包含了纽约市uber订单发生的日期、时间、经度和纬度信息.

表1 数据集

来源	数据集	I_1	I_2	I_3	I_4	NNZ
MovieLens	ratings-m1	6 K	4 K	5	/	1 M
MovieLens	ratings-m20	72 K	131 K	10	/	1 M
MovieLens+IMDb/Rotten Tomatoes	user_ratedmovies	72 K	65 K	10	/	856 K
Last.fm	user_taggedartists	2 K	19 K	13 K	/	186 K
Kaggle	Submissions	3 M	347 K	2	/	2 M
NYC Taxi & Limousine Commission (TLC)	Uber_Pickups	2 K	1 K	24	183	3 M

6.2 实验结果

为了分析多级并行SpTV算法在异构并行计算平台上的整体优化效果,图8展示了与串行SpTV运算相比,多级并行SpTV算法的并行加速比. 多级并行SpTV算法在所有测试数据集上平均获得了7.34倍并行加速比(最高在Uber_Pickups上获得10.60倍并行加速比,最低在ratings-m1上获得5.33倍并行加速比).

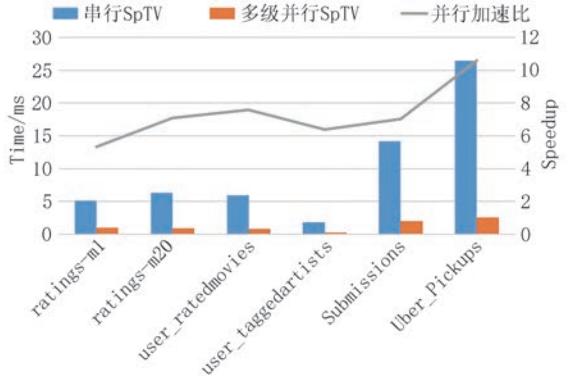

图8 多级并行SpTV算法的并行加速比

6.2.1 多维并行SpTV划分的评估

为了测试多级并行SpTV算法对并行计算节点数量的可扩展性,图10展示了算法在不同计算节点数量上的加速比. 实现结果显示,与采用1个计算节点相比,多级并行SpTV算法在8个计算节点上平均获得了3.57倍加速比,其中,在数据集user_taggedartists和Uber_Pickups上获得更高的加速比(分别为3.95倍和4.58倍),在Submissions上表现出较差的可扩展性(1.64倍加速比). 这是因为对于SpTV模n运算,面向节点级并行的$N-1$维张量划分不会在模n切面的维度对张量进行划分,即I_n的大小始终保持不变;同时,在每个计算节点中,CPU主存到GPU设备内存的传输数据都包括了长度为I_n的输入向量$\boldsymbol{v}$,在进行主存到设备内存的数据传输时,除了向量$\boldsymbol{v}$以外,其他数据的传输量都随着计算节点的增加而减少;因此,即使面向节点级并行的$N-1$维张量划分避免了节点间通信,但在每个计算节点内,对向量$\boldsymbol{v}$的数据传输开销是限制节点级并行可扩展性的主要原因. 根据表1中的数据,user_taggedartists和Uber_Pickups的I_n值很小(实验测试的是SpTV模1乘积,即$n=1$),即传输向量$\boldsymbol{v}$的开销较小,从而对节点级并行可扩展性的限制较小;然而Submissions的I_n值很大,传输向量$\boldsymbol{v}$的开销在总运算开销中的占比很大,因此限制了多级并行SpTV算法在该数据集上的节点级并行可扩展性.

为了研究面向GPU线程级并行的矩阵划分中删除空纤维操作对多级并行SpTV性能的影响,表2展示了删除空纤维操作对张量展开矩阵$\boldsymbol{X}_i$大小的影响,图9展示了删除空纤维操作对多级并行SpTV运行时间的影响. 删除空纤维操作能够使多级并行SpTV算法的运行时间平均减少43.61%,其中,在数据集user_taggedartists上的优化效果最佳(运行时间减少99.83%),在数据集ratings-m1上的优化效果最差(运行时间减少9.77%). 根据表2中的数据,删除空纤维操作缩减了展开矩阵$\boldsymbol{X}_i$在行向和列向两个维度的大小,$\boldsymbol{X}_i$的密度增加,非零元素分布的稀疏度降低. 一方面,行向维度大小(即I_n)的缩减能够相应地缩短向量$\boldsymbol{v}$的长度,同时,根据本文所采用的稀疏张量纤维压缩存储格式,列向维度大小(即M_i)的缩减能相应地减小数组$\boldsymbol{P}_f$的大小和结果张量的大小,从而减少了主机内存与设备内存之间的数据通信量. 另一方面,删除空纤维使得user_taggedartists展开矩阵的密度大大增加,非零元素能够更集中地分布在矩阵中,进而在进行多

⑤ https://www.nyc.gov/site/tlc/index.page/

维并行划分时，获得更好的并行负载均衡．因此，删除空纤维操作在数据集user_taggedartists上的优化效果最佳．同理，相比之下，删除空纤维操作在ratings-m1上的优化效果就没有那么明显．

为了测试多级并行SpTV对GPU并行线程数

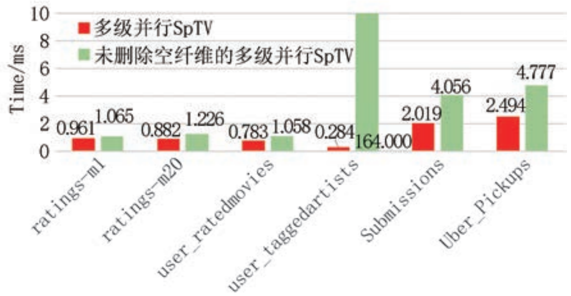

图9 面向GPU线程级并行的矩阵划分中删除空纤维操作对多级并行SpTV性能的影响

量的可扩展性，图11展示了GPU并行线程数量分别为1、4、16和64时算法的运行时间．由于每个GPU线程的计算任务相互独立，且没有数据通信，因此图11显示，多级并行SpTV在GPU线程上表现出良好的并行可扩展性．与采用1个GPU线程相比，采用64个GPU并行线程的多级并行SpTV算法的运行时间在6个测试数据集上平均获得了56.91倍的加速比．

6.2.2 多流并行异构优化策略的评估

图12比较了未采用基于张量块排序的多流并行技术、仅采用多流并行技术与采用了基于张量块排序的多流并行技术的多级并行SpTV算法的运行时间．实验结果证明，基于张量块排序的多流并行策略能够对多级并行SpTV产生良好的优化效果．与

表2 面向GPU线程级并行的矩阵划分中删除空纤维操作对张量展开矩阵X_i大小和密度的影响

数据集	$I_n \times M_i$（不删除空纤维）	$NNZ/(I_n \times M_i)$（不删除空纤维）	$I_n \times M_i$（删除空纤维）	$NNZ/(I_n \times M_i)$（删除空纤维）
ratings-m1	6040×19 760	8.38E-03	6040×16 912	9.79E-03
ratings-m20	7120×1 306 420	1.13E-04	7120×70 147	2.10E-03
user_ratedmovies	71 534×651 330	1.84E-05	71 534×63 015	1.90E-04
user_taggedartists	2100×23 705 536	3.75E-07	2100×109 750	8.09E-04
Submissions	3 253 292×693 430	7.08E-07	1 596 660×121 318	8.24E-06
Uber_Pickups	1717×5 006 880	3.85E-04	1717×692 597	2.78E-03

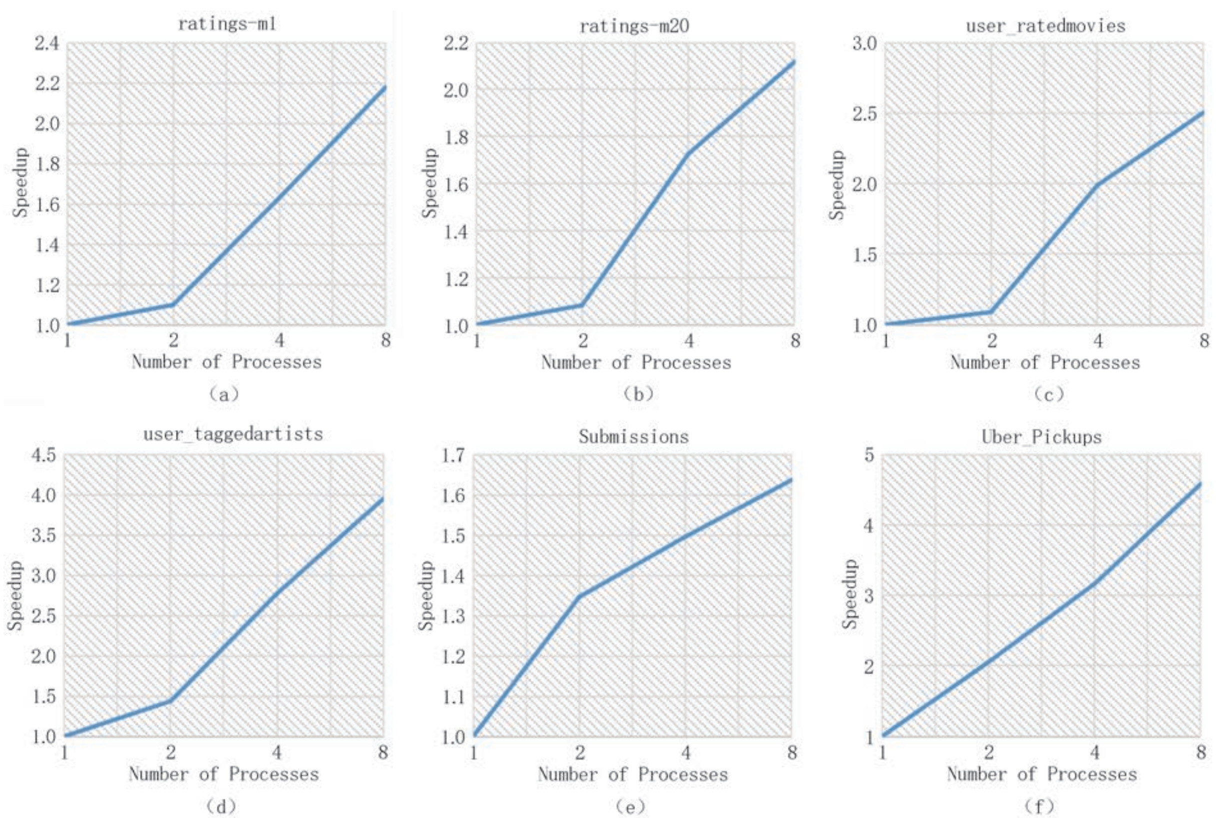

图10 多级并行的SpTV算法对计算节点数量的可扩展性

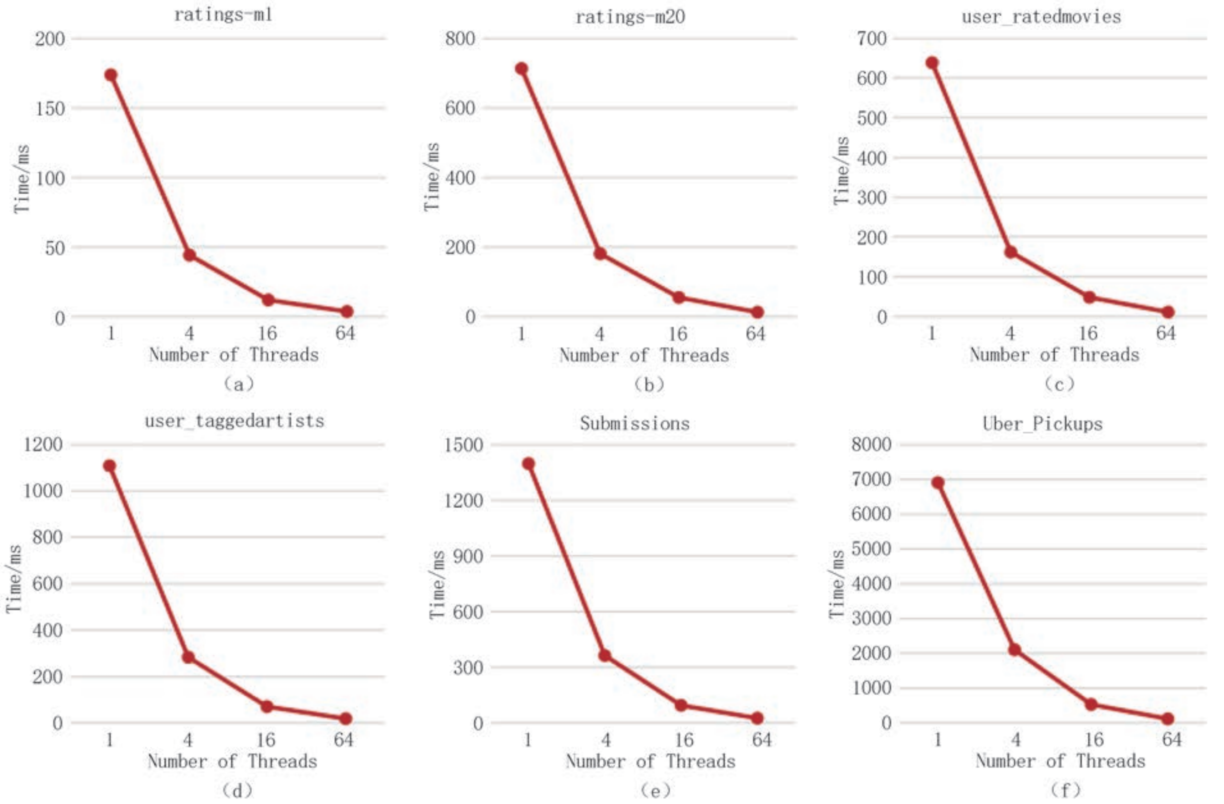

图11　多级并行的SpTV算法对GPU并行线程数量的可扩展性

未采用基于张量块排序的多流并行技术相比，仅采用多流并行技术的算法运行时间平均减少了40.09%．其中，在数据集user_taggedartists和Submissions上的优化效果较差，运行时间减少16.80%和27.14%．从图7中分析其原因，多流并行技术重叠了通信时间和并行计算时间，当相互重叠的通信时间与并行计算时间相差较小时，多流并行策略的重叠效果更好；相反，则优化效果更差．同时，根据算法设计，通信时间取决于I_n，M_i和NNZ的大小，计算时间取决于每个线程处理的非零元数量．如表2所示，对于user_taggedartists和Submissions，一方面，这两个数据集的I_n和M_i非常大，这导致通信时间在总计算时间里的占比很大；另一方面，这两个数据集的密度相对较小(删除空纤维后)，非零元素的分布更加稀疏，这导致每个线程处理的非零元数量较少，计算时间在总计算时间里的占比较小．这两方面的原因造成通信时间和计算时间相差较大，因此多流并行策略的优化效果较差．

与仅采用多流并行技术的算法相比，采用了基于张量块排序的多流并行技术的算法在运行时间上平均减少了17.48%．其中，在数据集Submissions上的优化效果最佳(运行时间减少29.65%)．根据表2中的数据，Submissions是所有数据集中非零元

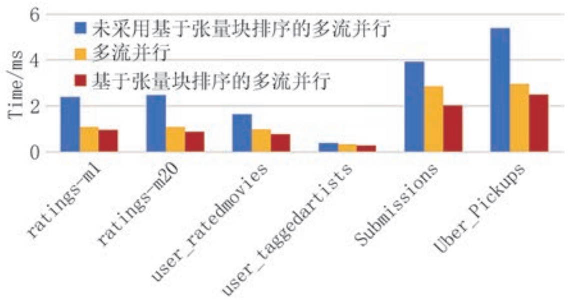

图12　基于张量块排序的多流并行策略的优化效果

素分布最稀疏的数据集(删除空纤维后)，稀疏度较大时，更可能发生各张量分块$sub\boldsymbol{X}_i^{(p)}$中非零元的数量相差较大、对各张量分块处理时间相差较大的情况．如图7所示，当各张量块$sub\boldsymbol{X}_i^{(p)}$的大小相差较大时，张量排序技术能够使较大张量块的计算、通信开销与多个较小张量块的计算、通信开销进行重叠，从而获得更好的优化效果．因此张量排序技术在Submissions上能表现出更好的优化效果．

6.2.3　与现有工作的对比

图13展示了多级并行的SpTV算法与cuSPARSE库⑥和aeSpTV[14]在6个测试数据集上的吞吐量对比．由于张量$\mathcal{X}$与向量$\boldsymbol{v}$的SpTV模n运算相当于

⑥ https://docs.nvidia.com/cuda/cusparse/

对张量模 n 展开矩阵的转置矩阵 $X_i^{(p)}$ 与 v 的稀疏矩阵乘向量(Sparse Matrix-Vector Multiplication, SpMV)运算，且cuSPARSE提供了一组用于处理稀疏矩阵基本线性代数的算法，由NVIDIA公司基于CUDA编程模型在GPU上设计与实现．因此，本实验将多级并行的SpTV运算与cuSPARSE库中提供的SpMV算法进行性能对比．

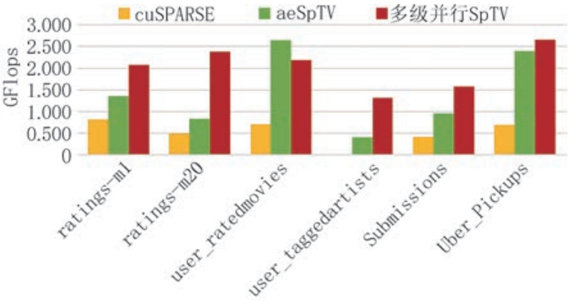

图13　多级并行SpTV算法与现有相关工作的吞吐量对比

如图13所示，由于多维并行SpTV划分方法和基于张量块排序的多流并行策略的优化作用，多级并行的SpTV算法在所有测试数据集上的运行效率都优于cuSPARSE库，特别是在数据集user_taggedartists上获得了超高的加速比，我们根据表2中的数据，分析其主要原因是多级并行SpTV算法中的空纤维删除操作极大地减小了user_taggedartists展开矩阵的列向维度(M_i)，同时也极大地减小了展开矩阵的列索引量、计算结果的大小以及主机内存与设备内存之间的通信量，因此与cuSPARSE库相比，多级并行SpTV算法在该数据集上表现出极大的性能优势．

与现有工作aeSpTV相比，多级并行SpTV算法的吞吐量平均获得1.88倍的加速比，但在数据集user_ratedmovies上，aeSpTV的吞吐量略高于多级并行SpTV算法，我们分析了可能的原因．第一，aeSpTV与多级并行SpTV的数据划分和存储方法不同，多级并行SpTV按张量展开矩阵的列向维度对非零元素进行并行任务划分和压缩存储，当想要利用更多GPU线程时，不需要额外增加索引量和通信量；然而aeSpTV按张量展开矩阵的行向维度对非零元素进行压缩存储，因此张量细粒度分块中非空行的数量影响了aeSpTV中的通信效率，当想要利用更多GPU线程时，需要对张量进行更细粒度的划分，随之而来的代价是细粒度分块中非空行的数量增多，增加了aeSpTV中的通信量；但对于数据集user_ratedmovies，随着GPU线程数的增加，aeSpTV并行计算效率的提升抵消了通信量增加所带来的负面影响．第二，aeSpTV采用了自适应张量划分选择器，选择最优的数据划分方法，从而获得更好的并行负载均衡．

7　总　结

本文基于CPU-GPU异构并行计算构架设计了多级并行的高效SpTV算法，为相关应用提供快速、高效的运算基础．首先，本文设计了一种多维并行SpTV划分方法，采用面向节点级并行的 N-1维张量划分和面向GPU线程级并行的矩阵划分，将SpTV运算映射到混合、多级并行的分布式CPU-GPU异构多/众核构架，充分利用计算节点间和节点内的多级并行计算能力．其次，基于这种多维划分方法，本文设计了一种基于稀疏张量纤维的压缩存储格式，优化SpTV运算的计算和访存模式．再次，本文设计了基于多流并行的异构高效SpTV算法，进一步采用稀疏张量的细粒度划分方法、多流并行运行机制和基于张量块排序的多流并行优化技术，对并行SpTV运算中的通信开销与计算开销进行相互隐藏，充分开发GPU的计算能力．在CPU-GPU构架上的实验证明，与现有相关工作aeSpTV相比，并行高效SpTV算法在所有测试数据集上最高获得3.28倍的加速比．

致　谢　感谢所有评审人员的建议和帮助!

参　考　文　献

[1] Poulenard A, Guibas L J. A functional approach to rotation equivariant non-linearities for tensor field networks//Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR2021), Virtual Event, 2021: 13174-13183

[2] Huang C. Ringcnn: Exploiting algebraically-sparse ring tensors for energy-efficient cnn-based computational imaging//Proceedings of the ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2021, Valencia, Spain, 2021: 1096-1109

[3] Zheng Y, Huang T, Zhao X, et al. Fully-connected tensor network decomposition and its application to higher-order tensor completion//Proceedings of the AAAI 2021 Conference on Artificial Intelligence, AAAI 2021 Conference on Innovative Applications of Artificial Intelligence, EAAI 2021, Symposium on Educational Advances in Artificial Intelligence, Virtual Event, 2021: 11071-11078

[4] Ioannidis V N, Zamzam A S, Giannakis G B, et al. Coupled graphs and tensor factorization for recommender systems and community detection. IEEE Transactions on Knowledge and Data Engineering, 2021, 33(3):909-920

[5] Kwon Y, Lee Y, Rhu M. Tensor casting: Co-designing algorithm-architecture for personalized recommendation training//Proceedings of the IEEE International Symposium on High-Performance Computer Architecture, HPCA 2021. Seoul, Republic of Korea, 2021: 235-248

[6] Chou S, Jang J R, Yang Y. Fast tensor factorization for large-scale context-aware recommendation from implicit feedback. IEEE Transactions on Big Data, 2020, 6(1):201-208

[7] Ho J C, Ghosh J, Sun J. Marble: high-throughput phenotyping from electronic health records via sparse nonnegative tensor factorization//Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD'14. New York, USA, 2014: 115-124

[8] Ren Y, Lou J, Xiong L, et al. Robust irregular tensor factorization and completion for temporal health data analysis//Proceedings of the ACM International Conference on Information and Knowledge Management. Ireland, 2020: 1295-1304

[9] Wang Y, Chen R, Ghosh J, et al. Rubik: Knowledge guided tensor factorization and completion for health data analytics//Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. Sydney, Australia, 2015: 1265-1274

[10] Kolda T G, Bader B W. Tensor decompositions and applications. SIAM Review, 2009, 51(3):455-500

[11] Ranadive T M, Baskaran M M. An all-at-once CP decomposition method for count tensors//Proceedings of the IEEE High Performance Extreme Computing Conference, HPEC 2021. Waltham, USA, 2021: 1-8

[12] Won T, Park I, Lee D, et al. Slicenstitch: Continuous CP decomposition of sparse tensor streams//Proceedings of the IEEE International Conference on Data Engineering, ICDE 2021. Chania, Greece, 2021: 816-827

[13] Karsavuran M O, Acer S, Aykanat C. Partitioning models for general medium-grain parallel sparse tensor decomposition. IEEE Transactions on Parallel Distributed Systems, 2021, 32(1):147-159

[14] Chen Y, Xiao G, Özsu M T, et al. aesptv: An adaptive and efficient framework for sparse tensor-vector product kernel on a high-performance computing platform. IEEE Transactions on Parallel Distributed Systems, 2020, 31(10):2329-2345

[15] Helal A E, Laukemann J, Checconi F, et al. ALTO: adaptive linearized storage of sparse tensors//Proceedings of the ACM International Conference on Supercomputing, ICS'21. USA, 2021: 404-416

[16] Xiao G, Li K, Chen Y, et al. Caspmv: A customized and accelerative spmv framework for the sunway taihulight. IEEE Transactions on Parallel Distributed Systems, 2021, 32(1): 131-146

[17] Qin E, Jeong G, Won W, et al. Extending sparse tensor accelerators to support multiple compression formats//Proceedings of the IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021. Portland, USA, 2021: 1014-1024

[18] Chen Y, li K, Yang W, et al. Performance-aware model for sparse matrix-matrix multiplication on the sunway taihulight supercomputer. IEEE Transactions on Parallel Distributed Systems, 2019, 30(4):923-938

[19] Bell N, Garland M. Implementing sparse matrix-vector multiplication on throughput-oriented processors//Proceedings of the ACM/IEEE Conference on High Performance Computing, SC'09. Portland, USA, 2009: 1-11

[20] Rhu M, O'connor M, Chatterjee N, et al. Compressing DMA engine: Leveraging activation sparsity for training deep neural networks//Proceedings of the IEEE International Symposium on High Performance Computer Architecture, HPCA 2018. Vienna, Austria, 2018: 78-91

[21] Li J, Sun J, Vuduc R W. Hicoo: hierarchical storage of sparse tensors//Proceedings of the International Conference for High Performance Computing, Networking, Storage, and Analysis, SC 2018, Dallas, USA, 2018: 19:1-19:15

[22] Smith S, Ravindran N, Sidiropoulos N D, et al. SPLATT: efficient and parallel sparse tensor-matrix multiplication//Proceedings of the IEEE International Parallel and Distributed Processing Symposium, IPDPS 2015. Hyderabad, India, 2015: 61-70

[23] Smith S, Karypis G. Tensor-matrix products with a compressed sparse tensor//Proceedings of the Workshop on Irregular Applications-Architectures and Algorithms, IA3 2015. Austin, USA, 2015: 5:1-5:7

[24] Abubaker N, Acer S, Aykanat C. True load balancing for matricized tensor times khatri-rao product. IEEE Transactions on Parallel Distributed Systems, 2021, 32(8):1974-1986

[25] Dun M, Li Y, Yang H, et al. An optimized tensor completion library for multiple gpus//Proceedings of the International Conference on Supercomputing, ICS'21. USA, 2021: 417-430

[26] Liu J, Ren J, Gioiosa R, et al. Sparta: high-performance, element-wise sparse tensor contraction on heterogeneous memory//Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP'21. Republic of Korea, 2021: 318-333

[27] Qiu Y, Zhou G, Zhang Y, et al. Canonical polyadic decomposition (CPD) of big tensors with low multilinear rank. Multimedia Tools and Applications, 2021, 80(15): 22987-23007

[28] Harper F M, Konstan J A. The movielens datasets: History and context. ACM Transactions on Interactive Intelligent Systems, 2016, 5(4):19:1-19:19

CHEN Yue-Dan, Ph. D., associate professor. Her research interests include high-performance computing, parallel and distributed processing, AI and big data computing.

XIAO Guo-Qing, Ph. D., professor. His research interests mainly include high-performance computing and AI computing.

YANG Wang-Dong, Ph. D., professor. His research interest mainly is parallel computing.

JIN Ji-Yong, M. S. D., assistant professor. His research interest mainly is distributed computing.

LONG Jun, Ph. D., professor. His research interests include big data computing and intelligent software systems.

LI Ken-Li, Ph. D., professor. His research interests include system software and applications for high-performance computing.

Background

Many applications give rise to multidimensional data that can be naturally represented via tensors. The tensors used in most real-world applications are extremely large and very sparse. The sparse tensor decomposition is an effective approach to predicting the unobserved data and is commonly used in machine learning, text analysis, healthcare analytics, and numerous other applications. SpTV is one of the most fundamental and time-intensive operations in computing tensor decomposition. Therefore, there have been a considerable number of researches on accelerating SpTV on heterogeneous parallel computing systems.

Parallelizing and accelerating large-scale SpTV on distributed CPU-GPU architectures faces three main challenges. First, how to effectively map SpTV operations to target distributed heterogeneous multi-core architectures and leverage the multi-level and hybrid parallelism. Second, how to compress the storage of sparse tensors effectively. The data structure determines the memory footprints and data access patterns in SpTV. Third, how to fully leverage the computing power of high-performance heterogeneous parallel computing systems for SpTV, based on the given task division method and sparse tensor compression storage format.

To alleviate the above-mentioned challenges, this paper exploits the hierarchical parallelism for SpTV on CPU-GPU heterogeneous parallel computing systems. First, we propose a multidimensional partitioning method to exploit the inter- and intra-node parallelism for SpTV. Second, based on the multidimensional data partitioning, we design a fiber-wise compressed storage format for sparse tensors to optimize the data access patterns for parallel SpTV. Third, we design the parallel streaming SpTV algorithm to overlap the data swapping cost and the computation overhead on GPUs.

This work has been supported in part by the Key-Area R&D Program of Guangdong Province (2021B0101190004), the Programs of the National Natural Science Foundation of China (No. 62172157, 62202149), the Programs of the Hunan Province (No. 2023GK2002、2021RC3062、2023JJ60002), the Programs of the Guangdong Province and Shenzhen (No. 2023A1515012915, JCYJ20210324135409026), and the Program of Zhejiang Lab (No. 2022RC0AB03).