计算机科学与探索
Journal of Frontiers of Computer Science and Technology
1673-9418/2024/18(04)-1094-15
doi: 10.3778/j.issn.1673-9418.2304039

TEB:GPU上矩阵分解重构的高效SpMV存储格式

王宇华,张宇琪,何俊飞,徐悦竹+,崔环宇
哈尔滨工程大学 计算机科学与技术学院,哈尔滨 150000
+通信作者 E-mail: xuyuezhu@hrbeu.edu.cn

摘　要:稀疏矩阵向量乘法(SpMV)是科学与工程领域中一个至关重要的计算过程,CSR(compressed sparse row)格式是最常用的稀疏矩阵存储格式之一,在图形处理器(GPU)平台上实现并行SpMV的过程中,其只存储稀疏矩阵的非零元,避免零元素填充所带来的计算冗余,节约存储空间,但存在着负载不均衡的问题,浪费了计算资源。针对上述问题,对近年来效果良好的存储格式进行了研究,提出了一种逐行分解重组存储格式——TEB(threshold-exchangeorder block)格式。该格式采用启发式阈值选择算法确定合适分割阈值,并结合基于重排序的行归并算法,对稀疏矩阵进行重构分解,使得块与块之间非零元个数尽可能得相近,其次结合CUDA(computer unified device architecture)线程技术,提出了基于TEB存储格式的子块间并行SpMV算法,能够合理分配计算资源,解决负载不均衡问题,从而提高SpMV并行计算效率。为了验证TEB存储格式的有效性,在NVIDIA Tesla V100平台上进行实验,结果表明TEB相较于PBC(partition-block-CSR)、AMF-CSR(adaptive multi-row folding of CSR)、CSR-Scalar(compressed sparse row-scalar)和CSR5(compressed sparse row 5)存储格式,在SpMV的时间性能方面平均可提升3.23、5.83、2.33和2.21倍;在浮点计算性能方面,平均可提高3.36、5.95、2.29和2.13倍。

关键词:稀疏矩阵向量乘法(SpMV);重新排序;CSR格式;负载均衡;存储格式;图形处理器(GPU)
文献标志码:A　　**中图分类号:**TP301

TEB: Efficient SpMV Storage Format for Matrix Decomposition and Reconstruction on GPU

WANG Yuhua, ZHANG Yuqi, HE Junfei, XU Yuezhu+, CUI Huanyu
School of Computer Science and Technology, Harbin Engineering University, Harbin 150000, China

Abstract: Sparse matrix-vector multiplication (SpMV) is a crucial computing process in the field of science and engineering. CSR (compressed sparse row) format is one of the most commonly used storage formats for sparse matrix. In the process of implementing parallel SpMV on the graphics processing unit (GPU), it only stores non-zero elements of sparse matrix, avoiding computational redundancy caused by zero element filling, and saving storage space. But there is a problem of load imbalance, which wastes computing resources. To address the aforementioned issues, storage formats with good performance in recent years have been studied, and a row by row decomposition and reorganization storage format—TEB (threshold-exchangeorder block) format has been proposed. The format first uses a heuristic threshold selection algorithm to determine the appropriate segmentation threshold, and combines the row merging algorithm based on reordering to reconstruct and decompose the sparse matrix, so that the number of non-zero elements between blocks is as close as possible. Furthermore, combined with CUDA (computer unified device

基金项目:国家自然科学基金(62072135)。
This work was supported by the National Natural Science Foundation of China (62072135).
收稿日期:2023-04-12　　**修回日期:**2023-07-27

architecture) thread technology, a parallel SpMV algorithm between sub blocks based on TEB storage format is proposed, which can reasonably allocate computing resources and solve the problem of load imbalance, thus improving the parallel computing efficiency of SpMV. In order to verify the effectiveness of the TEB storage format, experiments are conducted on the NVIDIA Tesla V100 platform. The results show that compared to PBC (partition-block-CSR), AMF-CSR (adaptive multi-row folding of CSR), CSR-Scalar (compressed sparse row-scalar), and CSR5 (compressed sparse row 5) storage formats, TEB can improve SpMV time performance by an average of 3.23×, 5.83×, 2.33×, and 2.21×. In terms of floating-point computing performance, the average improvement can be 3.36×, 5.95×, 2.29×, and 2.13×.

Key words: sparse matrix-vector multiplication (SpMV); reorder; compressed sparse row (CSR) format; load balancing; storage format; graphics processing unit (GPU)

求解大型线性方程组在工程和计算领域[1-3]的应用范围十分广泛，如电磁计算[4]、量子计算[5]、计算流体力学[6-7]、图计算[8]等。在求解过程中，SpMV（sparse matrix-vector multiplication）将大小为 $M\times N$ 维的稀疏矩阵 $\boldsymbol{A}$ 乘以 N 维稠密向量 $\boldsymbol{X}$，得到 M 维稠密向量 $\boldsymbol{Y}$ 的结果[9]。由于不同稀疏矩阵的稀疏度不同、矩阵的维数不同，影响SpMV计算效率的因素有很多。随着矩阵维数的不断增加，运算过程的数据量呈指数级增长模式，研究优化SpMV对于节约时间和内存压力具有实际意义，因此成为并行计算的研究热点之一。

各种稀疏矩阵在不同存储格式下，SpMV计算的并行效率也有很大差异。基于分块的存储格式会增强数据的空间和时间局部性[10]，利用局部性将相邻的非零元素一起加载到高速缓存中，从而减少内存访问的次数，提高计算速度。现有的GPU平台分块方法可以大致分为两类，一类是固定分块，另一类是按照稀疏矩阵非零元的分布结构进行分块。但是固定分块时，块与块之间的非零元素个数不一致，可能会导致不同线程块或者不同线程之间的工作负载不均衡，从而影响SpMV的并行效果。按照稀疏矩阵非零元的分布结构分块时，需要对稀疏矩阵结构进行详细分析并根据特定规则对其分块。不规则稀疏矩阵的非零元分布结构差异较大，研究合适的分块方法，是不规则稀疏矩阵并行计算的难点之一。

在GPU平台上合理分配计算资源，能够有效提高SpMV的并行效率。充分合理利用并行技术，使线程块间的运行时间大致相同，避免造成计算资源的空闲和浪费，是实现负载均衡的前提条件。因此，本文提出了一种新的存储格式TEB，采用启发式阈值选择算法和基于重排序的行归并算法将矩阵划分为大小相近的子块，从而实现负载均衡。

本文的主要贡献如下：

（1）提出了一种启发式阈值选择算法，通过对124个矩阵的三百多次实验结果分析，方差与阈值之间存在先减后增的规律，据此规律选择适应不同矩阵结构的阈值，以此作为划分矩阵块的理论依据。

（2）结合（1）中的阈值，提出了一种基于重排序的行归并算法来完成矩阵划分过程，使得GPU块与块之间非零元的个数具有最大相近性。

（3）基于（1）和（2）中的算法，本文提出了TEB（threshold-exchangeorder block）存储格式和基于TEB存储格式的SpMV并行算法，满足了并行计算过程中负载均衡的要求，同时经过佛罗里达大学公开数据集的实验证明，相较于PBC（partition-block-CSR）、AMF-CSR（adaptive multi-row folding of CSR）和CSR-Scalar（compressed sparse row-scalar），TEB可以有效提高SpMV的并行效率。

1 相关工作

稀疏矩阵非零元分布不均匀，可能会导致负载不均衡、计算冗余等问题，为了提高稀疏矩阵向量乘的性能，许多学者做了大量的研究。

Bian等人[11]提出的一种利用负载均衡和SIMD向量化有效处理SpMV的方法ALBUS（absolute load balancing using SIMD, single instruction multiple data），该方法将计算机需要处理的负载在各线程之间绝对平均地分配，并在各线程处理完毕后进行统一的边界数据处理，大幅减少了分块之间边界数据处理的次数。Liu等人[12]提出了CSR5格式，该格式将CSR中的列索引和值数组以转置的顺序分块存储，以便从连续的SIMD通道合并内存访问，通过两组额外的辅助信息来定位非零元行索引，具有较高的SIMD利用率。Zhang等人[13]提出了ACSR（aligned CSR）和AELL（aligned ELL）格式，该格式根据缓存行的宽度

合并相邻的列索引，不相邻的列索引之间需要补充零元素，能够提高SIMD单元的利用率。Namashivavam等人[10]提出的CVB（compressed vector blocks）格式通过分析稀疏矩阵非零元的分布来确定不规则稀疏矩阵块大小。采用多线程负载平衡策略和可变大小分块方案来适应不同的矩阵。Yesil等人[14]提出的LAV（locality-aware vectorization）方法将矩阵分为密集部分和稀疏部分，密集部分根据局部性进行格式化，而稀疏部分使用标准CSR算法进行处理，能够对SpMV的内存访问模式进行有效的向量化，解决了SIMD通道的负载均衡问题。Cui等人[15]提出的PBC格式采用一种自动获取最优块划分策略，使子块间非零元素数量的标准差最小，以满足子块间最大相近性，改善负载不平衡现象。Li等人[16]提出的VBSF（variable blocked-σ-SIMD format）格式将相邻的非零元素合并到可变长度的块中，以确保计算SpMV的访存连续性。Bian等人[17]提出的CSR2（compressed sparse row 2）方法利用SIMD向量化实现并行SpMV，其空间占用较少，只适用于CPU处理器。Gao等人[18]提出的AMF-CSR是一种自适应多行折叠算法，能够增加行块非零元素的密度，从而提高向量乘的访问局部性。Yang等人[19]提出了BCE（blocked stored format mixed CSR and ELL）格式，使用CSR和ELL混合格式来存储稀疏矩阵分块后的数据，基于稀疏矩阵非零元素分布的最优分块策略来提高SpMV的性能。

稀疏矩阵的结构和数据分布不均匀，不同处理单元之间的负载可能不平衡，导致一些处理单元的利用率较低，从而降低了整体的计算效率。采用分块的存储方式能够提高缓存的命中率，将子块合理分配给不同的处理单元，避免计算资源的浪费和负载不均衡的问题。由于分块过程中，难以避免存在辅助参数，现有的方法中大多根据经验设置参数，没有清晰地阐述设置过程及理由，而且现有的分块方式只能保证大部分子块的负载均衡，存在少数子块不均衡现象。因此，本文针对上述问题进行改进，提出了TEB存储格式和基于TEB的SpMV并行算法。

2 存储格式

2.1 CSR存储格式

定义1 假设稀疏矩阵 $\boldsymbol{A}$ 的大小为 $M\times N$，则 $\boldsymbol{A}$ 由 M 维行向量组成（如式（1）所示），可表示为 $\boldsymbol{R}_0,\boldsymbol{R}_1,\cdots,\boldsymbol{R}_{M-1}$，其中 $\boldsymbol{R}_i$ 为矩阵第 i 行，a 为单个非零元。

$$\boldsymbol{A}=\{\boldsymbol{R}_0,\boldsymbol{R}_1,\cdots,\boldsymbol{R}_{M-1}\}^{\mathrm{T}},\boldsymbol{R}_i=\{a_0,a_1,\cdots,a_{N-1}\} \tag{1}$$

CSR存储格式只存储每个非零项的索引，不考虑稀疏矩阵中非零元的分布，因此CSR适用于不规则稀疏矩阵的压缩。CSR存储格式采用三个数组来存储压缩后数据[20]，其中 *Values* 数组来存储稀疏矩阵中非零元的值，*Col* 数组来存储非零元的列索引，*Ptr* 数组来存储非零元的行偏移量[21]。根据以上三个数组，可以计算出非零元在稀疏矩阵的行索引、列索引和值，便于实现并行SpMV。

图1为8×8的稀疏矩阵 $\boldsymbol{A}$。本节利用CSR存储格式来存储矩阵 $\boldsymbol{A}$，其压缩后的数据如图2所示。

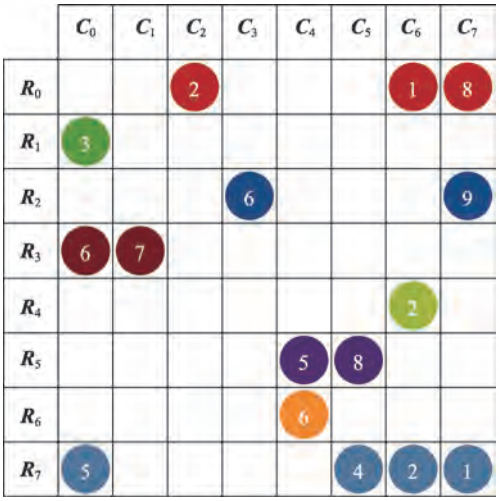


图1 8×8矩阵 *A*

Fig.1 8×8 matrix *A*

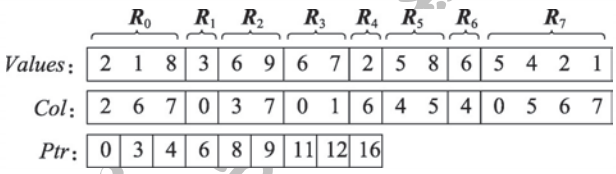


图2 CSR存储格式

Fig.2 CSR storage format

因为CSR只存储稀疏矩阵中非零元的值、列索引和行偏移量，所以CSR格式具有占用存储空间较少的优势[22]。但是，CSR是按照行主序的方式对非零元进行压缩存储，在数据的存储过程中，没有考虑到每行非零元个数的差异，可能会造成GPU工作负载不均衡，进而降低了SpMV的并行效率。

2.2 PBC存储格式

PBC存储格式采用一种重新排序策略对不同行的非零元进行组合，使不同块之间非零元的数量较

为均匀，尽可能得相同，通过遍历所有分块方式并计算块与块之间非零元个数标准差来选定标准差最小的分块方式。PBC采用五个数组存储稀疏矩阵，*Val_Data*存储排序后的值，*Dev_blockPos*存储了排序后的行索引，*Dev_w*存储了矩阵***A***每个块中最大非零元素个数的宽度，*Dev_p*存储了矩阵***A***中每个块的行偏移量，*Dev_offset*的值对应于矩阵***A***的列索引。

定义2 假设稀疏矩阵 ***A*** 可分为 P 个子块（如式(2)所示），则第 i 个子块表示为 $\boldsymbol{B}_i$。

$$\boldsymbol{A}=\bigcup_{i=1}^{P}\boldsymbol{B}_i,\boldsymbol{B}_i\cap\boldsymbol{B}_j=\varnothing(i\neq j) \tag{2}$$

采用PBC存储格式的分块方式如图3所示。

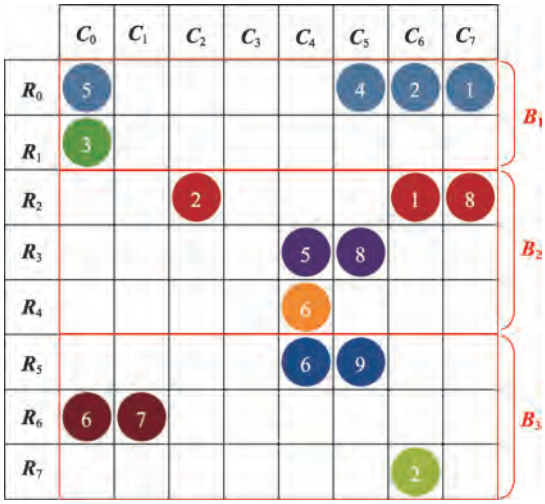


图3 PBC分块方式

Fig.3 PBC block method

PBC存储格式[15]将矩阵 ***A*** 分为3个子块，$\boldsymbol{B}_1$ 和 $\boldsymbol{B}_3$ 子块有5个非零元，$\boldsymbol{B}_2$ 子块有6个非零元，根据公式(5)计算 S^2_{PBC} 为0.23。本节利用PBC存储格式来存储矩阵 ***A***，其压缩后的数据如图4所示。

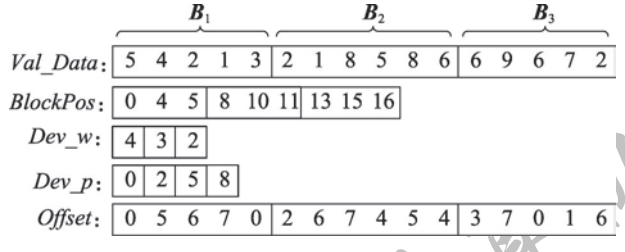


图4 PBC存储格式

Fig.4 PBC storage format

PBC存储格式具有良好的负载平衡能力，但由于存储一个稀疏矩阵需要对多种分块方式分别计算标准差，在预处理过程中，会带来较大的时间代价。

2.3 AMF-CSR存储格式

AMF-CSR存储格式采用自适应多行折叠算法来压缩稀疏矩阵，其根据矩阵的大小和非零元素的数量预先设置阈值 *Ts* 和 *Tl*[18]。非零元素总数小于 *Ts* 时，多行被折叠成一个行块。非零元素的总数小于 *Tl* 时，多个行块被折叠为一个子块。AMF-CSR采用三个数组存储稀疏矩阵，其中 *RowPtr* 由两部分组成。第一部分包括每个行块指针，第二部分包括每个子块的行块起始和结束索引。数组 *ColIndex* 和 *Val* 类似于固定的多行折叠存储，*ColIndex* 存储当前块中非零元的列索引，*Val* 存储非零元的值。

采用AMF-CSR存储格式的分块方式[18]（$Ts=4$，$Tl=7$ 为例）如图5所示。

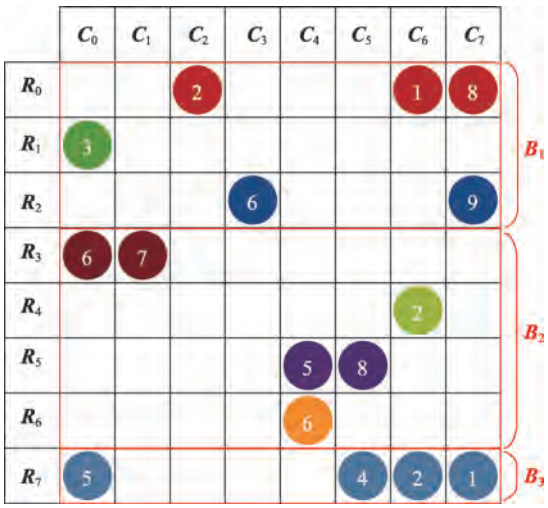


图5 AMF-CSR分块方式

Fig.5 AMF-CSR block method

AMF-CSR存储格式将矩阵 ***A*** 分为3个子块，$\boldsymbol{B}_1$ 和 $\boldsymbol{B}_2$ 子块有6个非零元，$\boldsymbol{B}_3$ 子块有4个非零元，根据式(5)计算 $S^2_{\mathrm{AMF-CSR}}$ 为1.29。本节利用AMF-CSR存储格式来存储矩阵 ***A***，其压缩后的数据如图6所示。

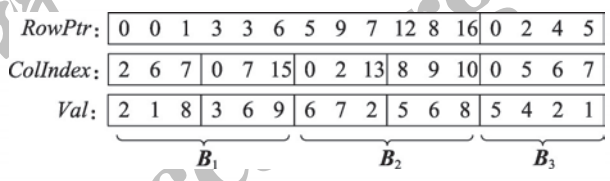


图6 AMF-CSR存储格式

Fig.6 AMF-CSR storage format

AMF-CSR存储格式根据预设定的阈值作为分块的依据，自适应地划分不同的稀疏矩阵，不仅提高了SpMV访问数据的局部性，而且在负载平衡方面也

有相应的提高,但部分阈值由经验设置,可能存在最后一块与其他块的非零元个数不平衡现象。

3 基于TEB存储格式的并行SpMV实现

3.1 TEB存储格式

针对现有的不规则稀疏矩阵,每行非零元素个数不均等、方差差异较大,容易导致负载不均衡等问题。本节提出了TEB存储格式,该格式首先采用启发式阈值选择算法,获得适应于当前矩阵非零元分布结构的分割阈值,然后在此分割阈值的基础上,采用基于重排序的行归并算法将稀疏矩阵分为大小相近的子块,使子块间的非零元数量具有最大相近性,从而解决负载不均衡问题。

3.1.1 启发式阈值选择算法

对于不规则稀疏矩阵来说,行与行之间非零元的数量差异较大,采用固定阈值的分块方式会造成计算资源不平衡,在块数确定的情况下,如果 *Threshold* 过大或者过小,都会造成部分块元素堆积,使得块与块之间非零元素数量不均衡,进而导致在并行阶段大多数线程处于空闲状态,只有部分线程还在并发执行,出现线程发散的现象。确定合理的分割阈值 *Threshold* ,将稀疏矩阵分为多个非零元数量相近的子块,满足了并行SpMV负载均衡的要求。

Threshold 的计算过程如式(3)所示。其中,AVG_{blN} 为 *blN* 个子块间非零元数量的均值,k 为 AVG_{blN} 的偏移参数。

$$Threshold_{blN} = AVG_{blN} \times k \tag{3}$$

式(4)进一步说明了 AVG_{blN} 的具体含义,其中,*NNZ* 为稀疏矩阵非零元总数,*blN* 为分割块数。*blN* 的取值范围为 $\{2,3,\cdots,M\}$。

$$AVG_{blN} = \frac{NNZ}{blN} \tag{4}$$

在 *blN* 不变的情况下,使用方差来衡量子块间非零元数量的偏离程度可以有效判断不同分割阈值的合理性,方差越小说明子块间非零元数量的差异越小,*Threshold* 越合理,找到使方差最小的 *Threshold* ,即找到当前块数的最优分块方式。

方差的计算公式如式(5)所示。其中,S^2_{blN} 为 *blN* 个子块间非零元数量的方差,NNZ_i 为第 i 个子块的非零元数量。

$$S^2_{blN} = \frac{\sum_{i=1}^{blN}(NNZ_i - AVG_{blN})^2}{blN} \tag{5}$$

在 *blN* 确定的情况下,AVG_{blN} 也随之确定,则 *Threshold* 和 k 之间存在线性关系。为了找到 *Threshold* 和 S^2_{blN} 的映射关系,对124个稀疏矩阵以0.005的增幅进行不同 k 值的实验。由于不同矩阵子块间非零元数量方差有很大差异,所以不能对纵坐标轴进行量化。因此,根据式(6)将不同 k 值下的所有方差归一化。其中,$\overline{S_i^2}$ 代表归一化后的方差,S_i^2 为实际方差,$\min(S^2)$ 和 $\max(S^2)$ 分别为最小方差和最大方差。

$$\overline{S_i^2} = \frac{S_i^2 - \min(S^2)}{\max(S^2) - \min(S^2)}, i = 1,2,\cdots \tag{6}$$

如下实验展示了 *blN* 为100、200和300时的实验结果。同时,为了更加清晰地描述不同 k 值下 S^2_{blN} 的趋势,每个 *blN* 提供了两组的实验结果,每组计算62个稀疏矩阵子块间方差的均值,作为此组的均方差。遍历各组矩阵在不同 k 值下的均方差,分析 k 与 S^2_{blN} 的总体趋势。

图7绘制了 *blN* 等于100时,k 与 S^2_{blN} 的关系图,通过对实验结果分析可知,随着 k 的稳步增长,在94%的稀疏矩阵中,S^2_{blN} 呈先减后增的趋势。96%的

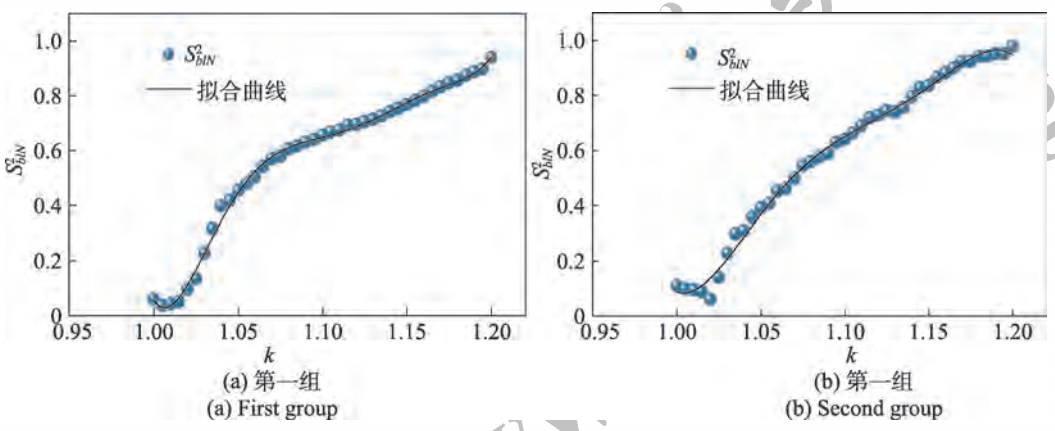


图7 不同 k 值与方差的总体趋势(*blN*=100)

Fig.7 Overall trend of different k values and variances (*blN*=100)

矩阵在 k 取1到1.03的范围内 *Threshold* 最优。

图8绘制了 *blN* 等于200时，k 与 S^2_{blN} 的关系图，通过分析实验结果可知，随着 k 值的稳步增长，在96%的稀疏矩阵中，S^2_{blN} 呈先减后增的趋势。80%的矩阵在 k 取1到1.03的范围内 *Threshold* 最优。

图9绘制了 *blN* 等于300时，k 值与 S^2_{blN} 的关系图，分析实验结果可知，随着 k 值的稳步增长，在94%的稀疏矩阵中，S^2_{blN} 呈先减后增的趋势。86%的矩阵在 k 取1到1.03的范围内 *Threshold* 最优。

根据 k 与 S^2_{blN} 的趋势图可知，S^2_{blN} 随着 k 的变化，存在一个拐点，使得 S^2_{blN} 最小，即存在一个最优 k，使得子块间非零元的个数具有最大相近性。通过对实验结果进行分析，发现在不同 *blN* 下取最优 *Threshold* 时的 k 不同，如何为 *blN* 选择合适的 k，是启发式阈值选择算法的重要研究内容。

基于上述实验，选取30个稀疏矩阵，分析最优 k 在 *blN* 等于100、200和300时的趋势。通过分析实验结果（如图10）可知，*blN* 和最优 k 之间存在线性关系，随着 *blN* 的增加，最优 k 值也随之增加。其中，94.44%的最优 k 值介于1到1.03之间。

依据实验结果的共同规律，设计了一种启发式阈值选择算法，该算法根据临界块和当前块平均非零元数量的关系，利用决策树选择合适的阈值，作为划分子块的依据。启发式阈值选择算法主要分为两大过程，首先计算临界块数和平均非零元数量，其次结合决策树完成阈值的选择。其具体流程如下：

首先，临界块（记为 *blN_critical* ）为最优 k 值从1变为1.01的最小块数，式（7）展示了临界块数的满足条件。

$$S^2_{k=1} > S^2_{k=1.01} \tag{7}$$

其中，$S^2_{k=1}$ 为 k 等于1时，子块间非零元数量的方差，$S^2_{k=1.01}$ 为 k 等于1.01时，子块间非零元数量的方差。

其次，利用式（4）计算块数为 *blN_critical* 时，子块的平均非零元个数，记为 *AVG_critical* 。

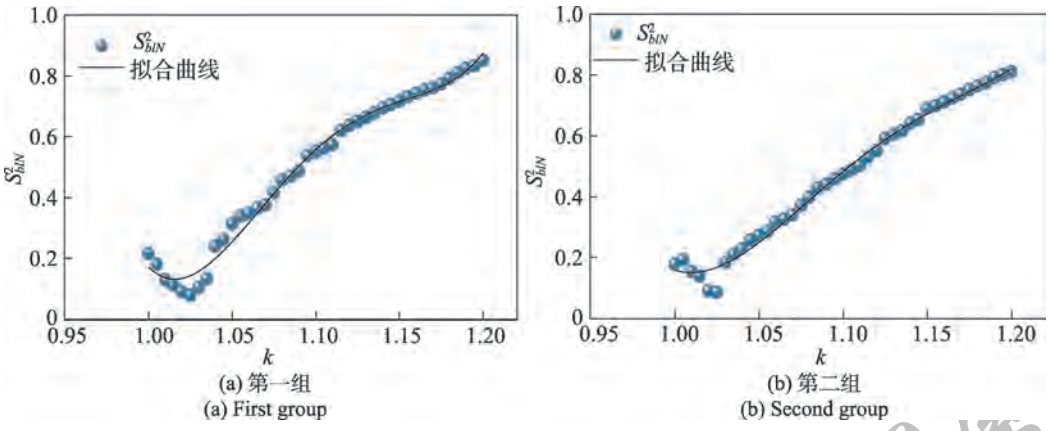


图8 不同 k 值与方差的总体趋势（*blN*=200）

Fig.8 Overall trend of different k values and variances (*blN*=200)

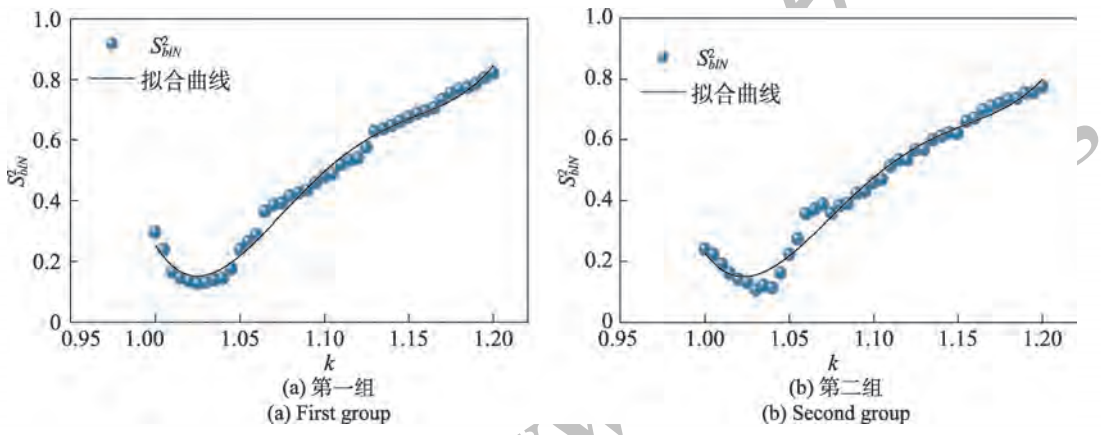


图9 不同 k 值与方差的总体趋势（*blN*=300）

Fig.9 Overall trend of different k values and variances (*blN*=300)

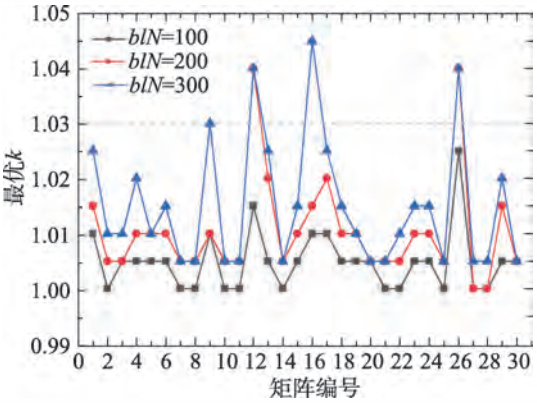


图10 最优 k 值与 blN 关系图趋势

Fig.10 Trend of relationship between optimal k value and blN

最后，根据阈值选择决策树来判断块数为 blN 时的最优 k 值，并根据式(3)计算出最优 *Threshold* 。阈值选择决策树如图11所示。

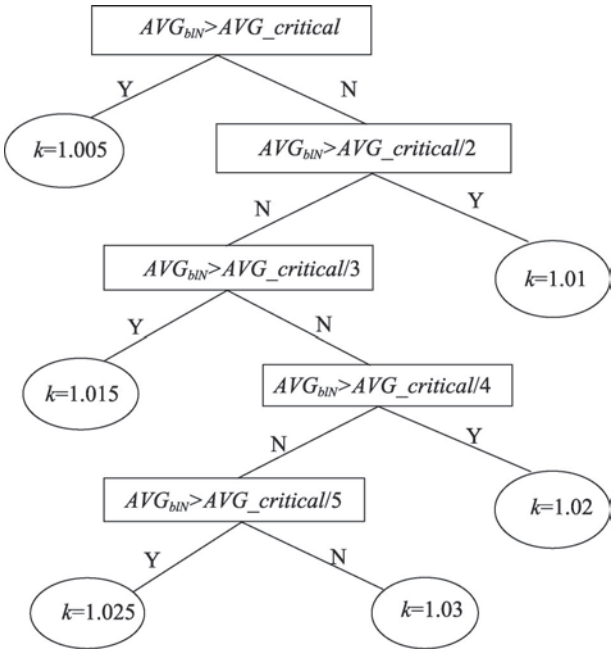


图11 阈值选择决策树

Fig.11 Threshold selection decision tree

在图11中，根据 AVG_{blN} 和 *AVG_critical* 确定 k 值。如果 $AVG_{blN} > AVG\_critical$ ，k 等于1.005；如果 $AVG_{blN} > AVG\_critical/2$ ，k 等于1.01；据此策略选取合适的 k ，如果所有条件都不满足，则 k 等于1.03。

算法1 启发式阈值选择算法

输入：每行非零元数 *rowNNZ*，*blN*，*NNZ*。

输出：最优阈值 *Threshold*。

1. $k_1 = 1$, $k_2 = 1.01$, *blN_critical*
2. for $i = 0$ to $blN-1$ do
3. *Threshold* = *NNZ*×*k* / *i*
4. 根据阈值划分矩阵
5. 分别计算 k_1 和 k_2 的 S^2
6. 判断 $S^2(k_1)$ 是否大于 $S^2(k_2)$
7. 如果是, *blN_critical* = *i*，退出循环
8. end for
9. *AVG_caitical* = *NNZ* / *blN_critical*
10. *AVG_blN* = *NNZ* / *blN*
11. 根据阈值选择决策树确定 *k*
12. *Threshold* = *AVG_blN*×*k*

启发式阈值选择算法的实现过程如算法1所示，其中，第3、4行分别计算 $k_1=1$ 和 $k_2=1.01$ 时的 *Threshold* ，并根据阈值对稀疏矩阵进行分割；第5~7行分别计算两种分块方式的方差，找到满足式(7)的 *blN_critical* ；第9、10行分别计算 *AVG_critical* 和 AVG_{blN} ；第11、12行根据 k 计算出最优 *Threshold* 。

启发式阈值选择算法的核心为确定合理分割阈值，为划分子块提供依据。根据此阈值划分后子块的非零元数量大致相同，满足了子块间非零元数量的最大相近性。

3.1.2 基于重排序的行归并算法

根据3.1.1小节中确定的最优分割阈值，结合重排序的行归并算法，对矩阵行进重排归并，找到最优的分块方式，使得块间非零元数量差异尽可能得小，从而满足负载平衡的要求。

定义3 假设存在数组 *RowNNZ_Sum*（简称 *RS* ）存储每行非零元素数量，*RS* 中有 M 个元素，$RS=\{RS_0,RS_1,\cdots,RS_{M-1}\}$。$RS_{max}$ 和 RS_{min} 分别表示数组 *RS* 中的最大值和最小值，用式(8)和(9)表示：

$$RS_{max} \geqslant RS_i, i=0,1,\cdots,M-1 \tag{8}$$

$$RS_{min} \leqslant RS_j, j=0,1,\cdots,M-1 \tag{9}$$

基于重排序的行归并算法流程如图12。

图12为基于重排序的行归并算法流程图，整个算法由两个嵌套循环组成。其中，外部循环为子块级循环，用于形成不同的子块，内部循环为行段级循环，用于合并子块中的行段。外部循环执行完成后，会生成当前矩阵的最优划分策略。

以矩阵 $\boldsymbol{A}$ 为例，$blN=4$，$k=1$ 时基于重排序的行归并算法划分过程如图13所示。由式(3)计算 *Threshold* =4。$\boldsymbol{A}$ 中每行的非零元个数为 *NNZ* ={3,1,2,2,1,2,1,4}，图13(a)将每行非零元个数排序后存入 *RS* 中，并创建 *S* 和 *Block* 数组，图13(b)将 *RS* 中最大元素4存入 *S* 中，由于4= *Threshold* ，将此行分为一个子块。图13(c)重置 *S* ，将 *RS* 中最大元素3存入 *S* 中，3< *Threshold* ，将 *RS* 中最小元素1存入 *S* ，3+1=

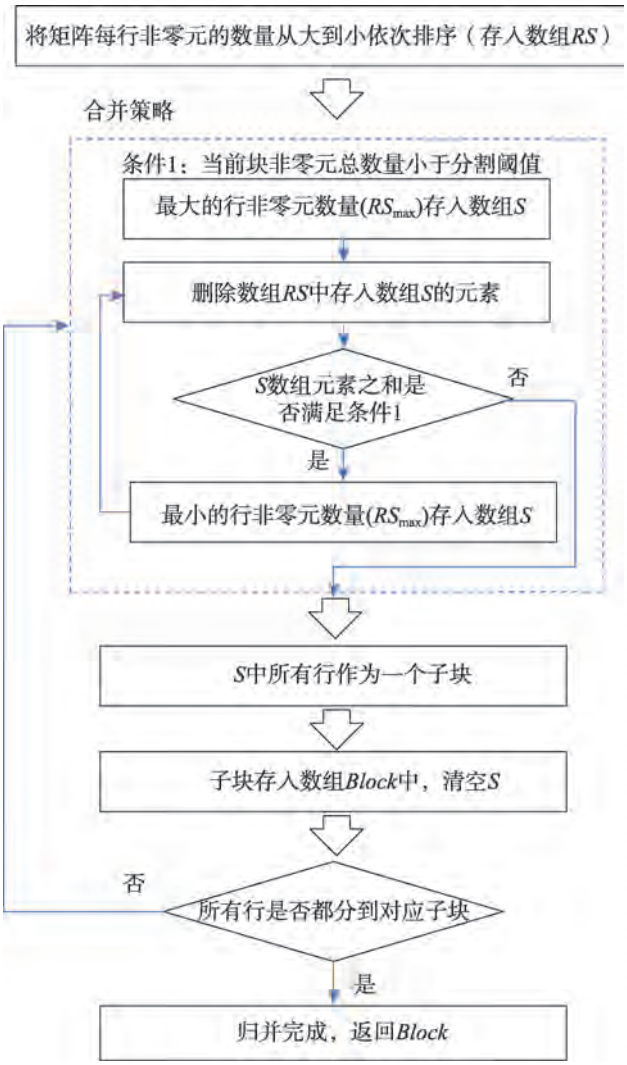


图12 基于重排序的行归并算法流程图

Fig.12 Flow chart of row merge algorithm based on reordering

Threshold，将以上行段归并为一个子块。图13(d)重置 *S*，将 *RS* 中最大元素2存入 *S* 中，不满足 *Threshold* 条件，继续归并最小行，将2，1，1分为一个子块。图13(e)将 *RS* 中最大元素2存入 *S* 中，继续归并最小行，将2，2归并为一个子块。图13(f)表示矩阵 ***A*** 的最终行归并结果。

算法2 基于重排序的行归并算法

输入：每行非零元数 *rowNNZ*，*blN*，*NNZ*，*Threshold*，*N*。

输出：划分结果 *Blo_Idx*，*RowNNZ_Sum*。

1. *begin* = 0，*end* = *N*−1，*count* = 0
2. for *i* = 0 to *blN*−1 do
3. *SUM* = *RowNNZ_Sum*[*begin*]
4. *count*++
5. while *SUM* + *rowNNZ*[*end*] < *Threshold* do

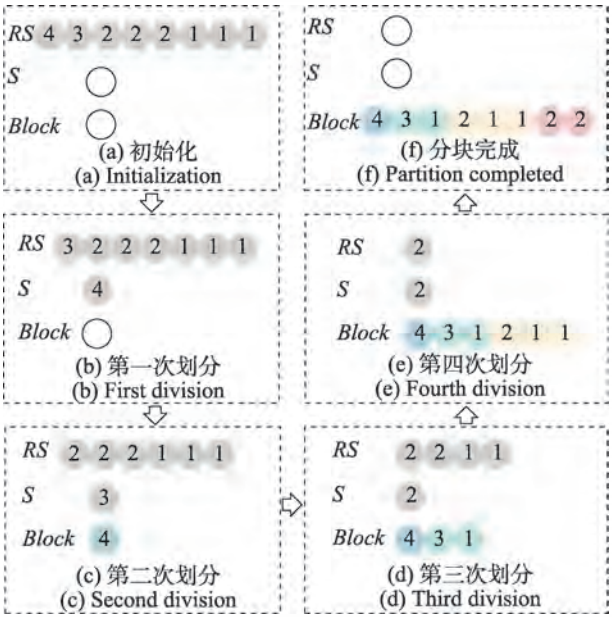


图13 基于重排序的行归并算法示例

Fig.13 Example of merge algorithm based on reordering rows

6. *SUM* += *rowNNZ*[*end*]
7. *count*++，*end*−−
8. end while
9. *begin*++
10. *Blo_Idx*[*i*+1] = *count*
11. *RowNNZ_Sum*[*i*] = *SUM*
12. end for
13. *Blo_Idx*[*blN*] = *N*
14. *RowNNZ_Sum*[*blN*] = *NNZ*

基于重排序的行归并算法具体实现过程如算法2所示。其中，第2~12行为外部循环阶段；第5~8行为内部循环阶段；第13、14行记录最后一块的数据。

TEB存储格式采用启发式阈值选择算法和基于重排序的行归并算法，以行主序的方式将不同行归并为非零元数量相近的子块。以矩阵 ***A*** 为例，其划分结果如图14所示。TEB格式将 ***A*** 分为四个子块，每个子块都有4个非零元，根据式(5)计算 S^2_{TEB} 为0，相较于PBC和AMF-CSR格式，使用TEB格式时，子块间非零元数量更加均衡，相似性更高。

遍历 *blN*，找到使 S^2_{blN} 最小的 *blN*，作为该矩阵的最终分割块数。但需要特别注意的是，在此过程中，并不需要遍历所有块数，如果满足式(10)，则不需要遍历大于 *blN* 的分块方式，因为满足式(10)的 S^2_{blN} 较大，子块间的相似性小。如此，便可以有效地缩短在CPU平台上矩阵压缩的执行时间。

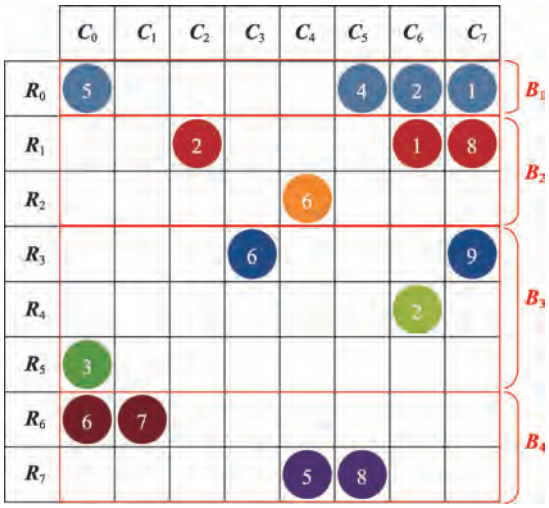


图14 TEB存储格式划分过程

Fig.14 TEB storage format division process

$$Row\_NNZNum[0] > 2 \times Threshold_{blN} \tag{10}$$

其中，$Row\_NNZNum[0]$ 为最大行非零元数量，$Threshold_{blN}$ 为 blN 块数的分割阈值。

TEB格式采用四个数组存储分块后的压缩数据。其中 *Values* 和 *Col_Idx* 数组存储重新排序后的非零元值和列索引，*Blo_Idx* 数组用于存储子块的行偏移量，*RowNNZ_Sum* 数组用于存储每行非零元数量的偏移量。此外，在重新排序过程中需要记录每行的初始索引，保证计算结果的正确性。利用TEB存储格式来存储矩阵 ***A***，其压缩后的数据如图15所示。

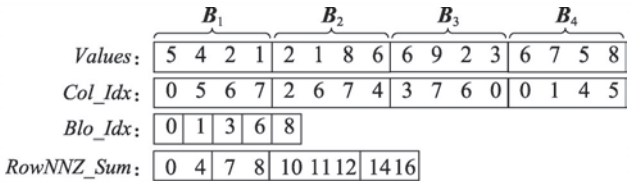


图15 TEB存储格式

Fig.15 TEB storage format

3.2 并行SpMV算法实现

对于不规则稀疏矩阵来说，每行非零元个数差异较大，导致GPU工作负载不均衡。在3.1节中提出的TEB存储格式，可以有效平衡子块间非零元的数量。本节基于TEB存储格式，在GPU平台上进行SpMV的并行运算。其中，每个子块对应于一个线程块，满足块与块之间的负载均衡条件，能够减少线程间的发散，有效提高SPMV的并行计算效率。

基于TEB存储格式的并行SpMV策略首先将并行计算过程中所需的数组从主机端传递到设备端，设备端通过对数组进行索引，确定每个线程块对应的数据（如图16所示）。然后线程块并行执行SpMV算法，最后将计算结果 ***Y*** 从设备端传到主机端。

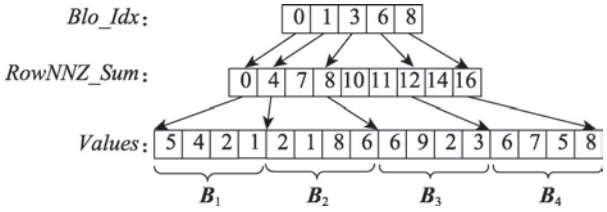


图16 索引数据

Fig.16 Index data

3.1.2小节中提出的基于重排序的行归并算法将矩阵 ***A*** 分为四个子块，分别对应于四个线程块（如图17所示），每个线程块索引 *Values* 和其对应的 ***X*** 值，采用并行的方式对线程块中的数据进行SpMV运算，并将最终结果存到 ***Y*** 数组中。实现了四个线程块并发执行，SpMV的运算时间也大致相同，有效平衡了GPU工作负载。

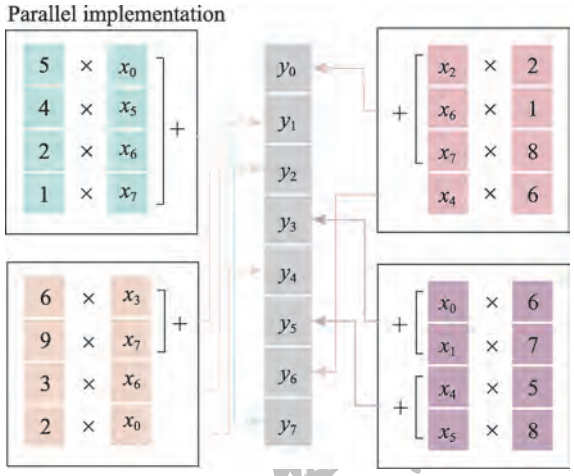


图17 线程块实现并行SpMV

Fig.17 Thread block implementing parallel SpMV

算法3 基于TEB存储格式的并行SpMV算法

输入：*Values*，*Col_Idx*，*Blo_Idx*，*RowNNZ_Sum*，***x***。

输出：***Result***。

1. *blockID* = *blockIdx.x*，*threadID* = *threadIdx.x*
2. *row* = *Blo_Idx*[*blockID*] + *threadID*
3. *start* = *RowNNZ_Sum*[*row*]
4. *end* = *RowNNZ_Sum*[*row*+1]
5. *mid*=*Blo_Idx*[*blockID*+1]−*Blo_Idx*[*blockID*]
6. *sum* = 0
7. if *threadID* < *mid* then
8. for *i* = *start* to *end* do
9. *sum* += *Values*[*i*] * ***x***[*Col_Idx*]

10. end for
11. ***Result***[*row*] = *sum*
12. end if

基于TEB存储格式的并行SpMV算法（如算法3所示）的第1行表示当前的线程块ID和每个块中的线程ID，第2行用于定位不同块中的各行索引，第3、4行用于定位此线程所处理的非零元开始和结束的索引，第5行表示当前块中非零元的行数，第7行的判断用于限制当前线程块中线程的个数，避免出现空闲线程等待的情况，第8~11行是核心步骤，用于计算SpMV，并将结果存储在对应的数组。

本节提出的基于TEB存储格式的并行SpMV算法，能够给每个矩阵子块合理分配线程块和线程，解决了负载不均衡问题，有效避免了计算资源的空闲和浪费，增强了并行SpMV的时间、空间性能，同时提升了并行计算的效率。

4 实验分析

第3章提出的基于TEB存储格式的并行SpMV算法解决了并行过程中的负载不均衡问题，本章从运行时间、加速比和预处理时间等方面分析了算法的实验结果，评估了其实验性能，验证了所提算法的有效性。

4.1 实验设置

本实验所采用的数据全部来自佛罗里达大学公开的稀疏矩阵数据集（SuiteSparse矩阵集合），表1中列出了20个用于实验的稀疏矩阵，它们来自各种领域，如电网问题、流体力学问题、材料问题、声学问题、2D/3D问题等领域。表中展示了矩阵的维度，大小以及分布情况。其中，有11个规则矩阵，9个不规则矩阵。

本实验在NVIDIA Tesla V100S-PCIE-32GB显卡上进行，有5 120个流处理单元，核心频率为1 245 MHz，核心架构为Volta，有80个SM，线程块的最大线程数量为1 024，FP32的性能为16.35 TFLOPS。CPU版本为Intel® Xeon® Gold 6240R CPU @ 2.40 GHz。服务器操作系统为CentOS-7。

4.2 预处理时间分析

预处理过程是准备并行SpMV所需数据的阶段，可以看作CSR格式转换为TEB格式的耗时，由于TEB存储格式的实现分为两个主要阶段，启发式阈值选择阶段和行重排归并阶段，记录两个阶段的耗时，并分析预处理过程的时间开销（*PreTime*）。

在float和double两种数据类型下，首先分析不同矩阵在启发式阈值选择阶段的耗时，由图18可知，矩阵在float数据类型的耗时均小于0.6 ms，在double数据类型的耗时均小于0.7 ms。除fd18外，单精度的耗时都略小于双精度的耗时，数据类型对启发式阈值选择阶段的耗时有所影响，但因其耗时较短，影响效果不明显。

行重排归并阶段由于需要将所有行依次分到相应的子块中，所以此阶段耗时要大于启发式阈值选择阶段。图19分析了不同矩阵在行归并重排阶段的耗时。由图可知，float和double数据类型的差异在此阶段表现不明显，除vibrobox和c-64的单精度耗时要小于双精度耗时外，其他矩阵的两种数据类型耗时极为相似。分析实验结果可知，矩阵在0.6 s内完成重排归并，划分相应的子块。维度小于5 000的矩阵在阈值选择阶段和行重排归并阶段的耗时相近，因此预处理过程能够极快地完成。

由以上分析可知，不同数据类型在预处理阶段的差异不明显，且两个阶段的耗时趋势大致相同。预处理阶段的耗时主要来自于行重排归并阶段，综合考虑矩阵维度、非零元数量及分布情况可知，TEB存储格式的预处理时间在可以接受的范围内。

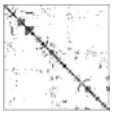
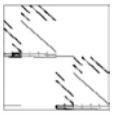
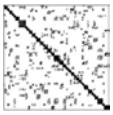
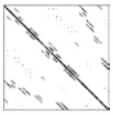
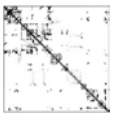
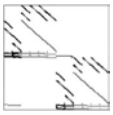
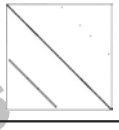
4.3 SpMV性能分析

针对基于TEB存储格式的并行SpMV算法进行性能分析，主要分为两个方面，分析SpMV运行时间和浮点运算性能。选择目前较为新颖和有代表性的分块存储格式CSR-Scalar[21]、CSR5、PBC和AMF-CSR与TEB存储格式做对比实验，分别计算五种格式SpMV的运行时间，以此作为比较时间性能的依据，因为运行一次SpMV的时间极短且存在一定的误差，所以记录每种格式运行10 000次SpMV的耗时作为本实验SpMV的运行时间（*STime*），以此来消除单次运行时间的不准确性，使SpMV运行时间的实验结果误差最小化。

为了观察不同数据类型对SpMV运行时间的影响，首先在float数据类型下，记录五种存储格式在规则矩阵上SpMV的运行时间（如图20所示），实验结果与时间性能成反比，运行结果耗时越少，时间性能越高。分析实验结果，除矩阵5外，TEB的运行时间均短于其他格式，矩阵5上TEB的*STime*约为CSR-Scalar的1.58倍，CSR-Scalar在五种格式中耗时最少，由于矩阵5每行非零元数量差异较小，相比于其他矩阵来说更均衡，所以CSR-Scalar格式的运行时间较短。PBC在矩阵8和11的耗时超出1 s，剩余其他格式都在1 s内完成10 000次SpMV。相比于AMF-

表1 稀疏矩阵

Table 1 Sparse matrix

编号	名称	维数/10^3	非零元/10^3	分布	编号	名称	维数/10^3	非零元/10^3	分布
1	1138_bus	1.13	4.05		11	e40r0000	17.28	553.96	
2	nasa1824	1.92	39.21		12	cavity10	2.60	76.17	
3	vibrobox	12.33	301.70		13	ex11	16.61	1 096.95	
4	s1rmq4m1	5.49	262.41		14	igbt3	10.94	130.50	
5	finan512	74.75	596.99		15	fd18	16.43	63.41	
6	nd2010	133.77	625.95		16	torso2	115.97	1 033.47	
7	c-64	51.04	707.99		17	af23560	23.56	460.60	
8	ct20stif	52.33	2 600.30		18	fidap015	6.87	96.42	
9	crystm03	24.70	583.77		19	fd15	11.53	44.21	
10	bcsstk25	15.44	252.24		20	dw8192	8.19	41.75	

CSR、PBC、CSR-Scalar和CSR5，TEB时间性能平均提升2.92、6.4、1.79和2.46倍，最高提升5.13、11.64、3.66和4.22倍。

图21展示了float数据类型下，五种存储格式在不规则矩阵上的*STime*。TEB的运行时间均优于其他格式，除矩阵13、16外，其他矩阵TEB格式的*STime*都在0.1 s内。13和16在所有不规则矩阵中的耗时最多，由于其非零元数量较大，所以运行时间相对较长，规则矩阵8与不规则矩阵13的非零元数量在一个量级，但其耗时的差别不明显，对于矩阵8和13来说，非零元分布的规则性对SpMV运行时间的影响较小。相比于AMF-CSR、PBC、CSR-Scalar和CSR5，TEB时间性能平均提升3.42、4.67、2.35和2.27倍，最高提升为5.18、9.72、5.43和3.71倍。

为了更加清晰地分析各种格式的运行时间，根据式(11)，计算TEB在AMF-CSR、PBC、CSR-Scalar和CSR5上的加速比。其中，*SpeedUp*为加速比，$STime_i$为每种格式的SpMV运行时间，i取不同的格式。

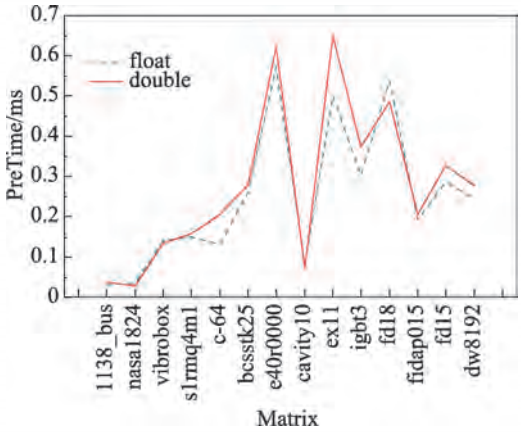


图18 启发式阈值选择阶段耗时

Fig.18 Time consumption of heuristic threshold selection phase

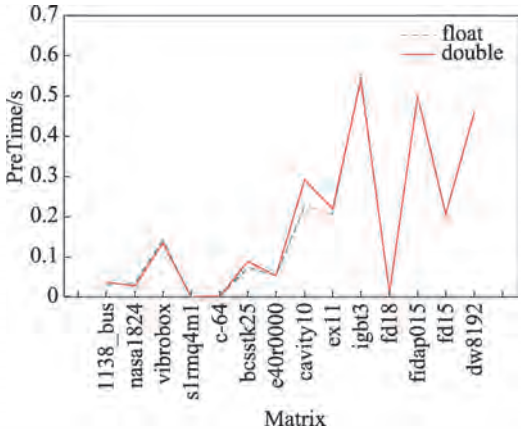


图19 行重排归并阶段耗时

Fig.19 Time consumption of row rearrangement and merging stage

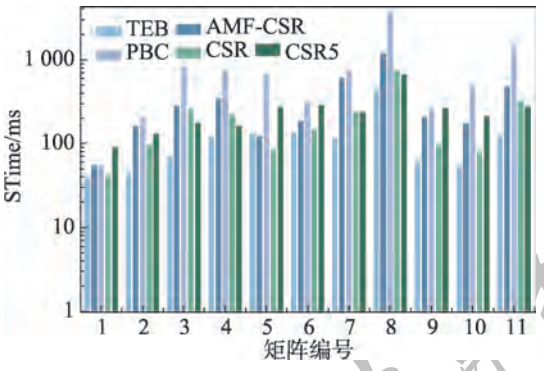


图20 规则矩阵SpMV执行时间(float)

Fig.20 Regular matrix SpMV execution time (float)

$$SpeedUp = \frac{STime_i}{STime_{\text{TEB}}},\quad i = \text{AMF-CSR, PBC, CSR, CSR5} \tag{11}$$

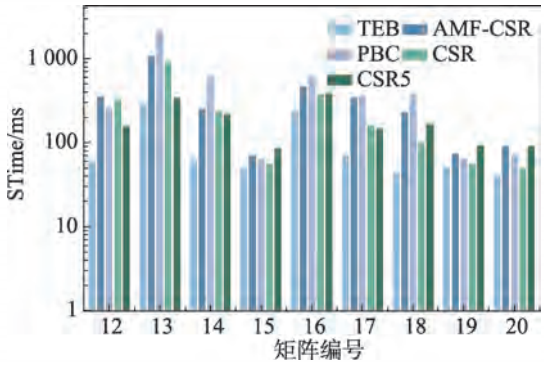


图21 不规则矩阵SpMV执行时间(float)

Fig.21 Irregular matrix SpMV execution time (float)

分析实验结果可知(如图22所示),在float数据类型下,由于矩阵5的TEB运行时间要略高于其他两种格式,所以AMF-CSR和CSR-Scalar格式的加速比分别为0.94和0.63,除此之外,其余的加速比均大于1。对比PBC格式,TEB表现出了良好的运行时间优势,由于矩阵3非零元分散且数量多,最高加速比可达11.64倍。相较于AMF-CSR、PBC、CSR-Scalar和CSR5,TEB的平均加速比分别提升3.15、5.62、2.04和2.38倍。

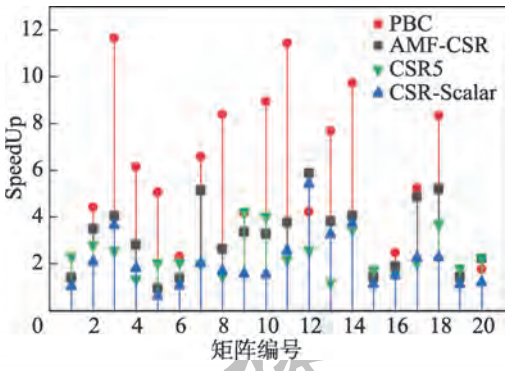


图22 不同格式的SpMV执行时间加速比(float)

Fig.22 SpMV execution time acceleration ratio (float) for different formats

由于SpMV的计算分为两个部分,首先每个非零元与对应 $\boldsymbol{X}$ 向量相乘,之后每行进行累加,运算过程中进行了 NNZ 次乘法运算, $NNZ-M$ 次加法运算,则SpMV的运算数为 $2\times NNZ-M$ 。为了分析五种格式在GPU服务器的浮点运算性能,采用每秒10亿次的浮点运算数来衡量GPU内核的性能,根据式(12),计算浮点运算性能,其中 NNZ 为非零元总数, M 为矩阵维度, $STime_i$ 为每种格式的SpMV运行时间。图23分析了不同存储格式的浮点运算性能,由实验结果可知,对比其他格式,除矩阵5之外,TEB都表现出了

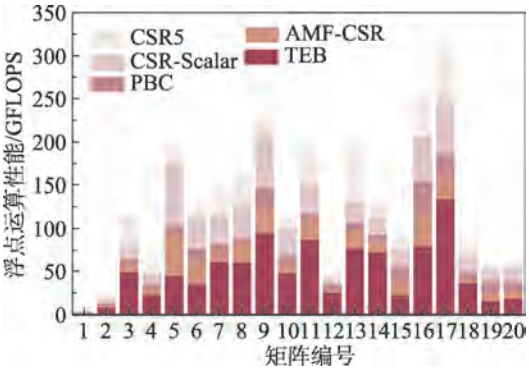


图23 不同存储格式浮点运算性能(float)

Fig.23 Performance of floating-point operations in different storage formats (float)

较高的单精度浮点运算性能,最高可达134 GFLOPS。由于内核性能与SpMV运行时间成反比,所以TEB在矩阵5的表现欠佳,为45.1 GFLOPS。相较于AMF-CSR、PBC、CSR-Scalar和CSR5,TEB的浮点运算性能分别提升3.25、5.58、2.03和2.71倍。

$$浮点运算性能=\frac{2\times NNZ-M}{STime_i} \tag{12}$$

在double数据类型下,记录五种存储格式在规则矩阵上SpMV的运行时间(如图24所示)。除矩阵5和6外,TEB的运行时间均优于其他格式。矩阵5的实验结果与float数据类型的结果类似,CSR-Scalar格式的运行时间最短,为96.05 ms。此外,矩阵6的CSR-Scalar运行时间也优于TEB,由于TEB格式float变为double类型转换时间为45 ms,而CSR-Scalar的数据类型转换时间为26 ms,TEB的转换时间较长且非零元数量较多,在运算过程中TEB的转换时间累加,进而导致其SpMV运行时间有所增加。大部分矩阵double相较于float数据类型,运行时间都有不同幅度的增加。相比于AMF-CSR、PBC、CSR-Scalar和CSR5,TEB时间性能平均提升3.07、6.42、2.1和2.16倍,最高提升4.7、13.42、3.55和3.82倍。

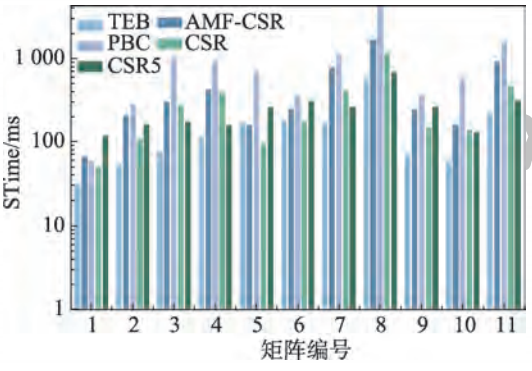


图24 规则矩阵SpMV执行时间(double)

Fig.24 Regular matrix SpMV execution time (double)

图25展示了五种存储格式在不规则矩阵上SpMV的运行时间。在double数据类型下,TEB的运行时间仍然较短,没有出现因数据类型转换时间较长导致运行时间增加的情况。PBC和AMF-CSR格式在矩阵16的数据类型转换时间分别为429 ms和907 ms,由于其非零元数量较多,double类型对其运行时间的影响较大。对比float类型,double的运行时间有所增加,但总体趋势没有发生变化。相比于AMF-CSR、PBC、CSR-Scalar和CSR5,TEB时间性能平均提升3.62、5.57、3.03和1.86倍。运行时间最高提升6.4、12.94、6.15和2.32倍。

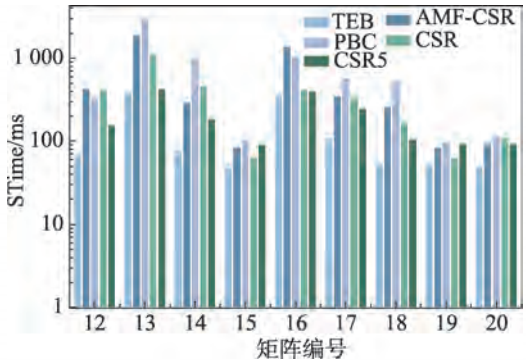


图25 不规则矩阵SpMV执行时间(double)

Fig.25 Irregular matrix SpMV execution time (double)

为了更加清楚地分析SpMV运行时间在双精度数据类型的提升效果,绘制了TEB在AMF-CSR、PBC、CSR-Scalar和CSR5上的SpMV执行时间加速比(如图26所示),在double数据类型下,TEB相对于其他存储格式(除矩阵5外)的加速比均大于1,对比PBC格式,TEB表现出了良好的运行时间优势,最高加速比可达13.42倍。矩阵5的AMF-CSR和CSR-

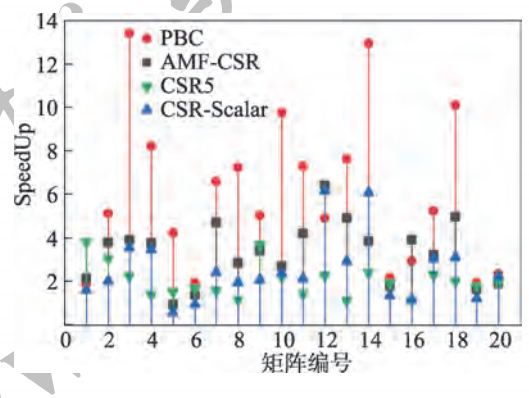


图26 不同格式的SpMV执行时间加速比(double)

Fig.26 SpMV execution time acceleration ratio (double) for different formats

Scalar格式的运行时间较短，加速比分别为0.92和0.57。矩阵6的CSR-Scalar格式运行时间较短，加速比为0.94。相较于AMF-CSR、PBC、CSR-Scalar和CSR5，TEB的平均加速比分别提升3.31、6.04、2.51和2.03倍。

图27展示了五种格式的双精度浮点运算性能，根据式(10)，计算出每种存储格式的性能。分析使用结果可知，TEB表现出了较好的双精度浮点运算性能，平均运算性能可达39.95 GFLOPS，矩阵9和17的运算性能在所有矩阵中较高，都达到了80以上。由SpMV运行时间可知，矩阵5和6的CSR-Scalar存储格式耗时短，所以TEB的浮点运算性能略低于CSR-Scalar。AMF-CSR、PBC、CSR-Scalar和CSR5的平均浮点运算性能分别为13.14、7.61、20.08和22.3 GFLOPS，TEB相较于其他格式的浮点运算性能分别提升2.46、5.31、1.52和1.08倍。TEB的运行时间和浮点运算性能总体提升情况如表2所示。

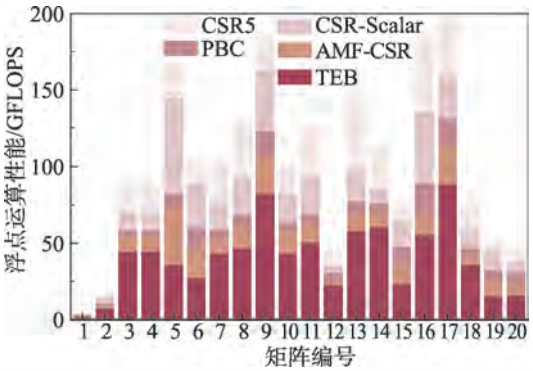


图27 不同存储格式浮点运算性能(double)

Fig.27 Performance of floating-point operations in different storage formats (double)

表2 TEB运行时间和浮点运算性能平均加速比

Table 2 TEB running time and floating-point arithmetic performance average speedup

存储格式	SpMV运行时间/ms	浮点运算性能/GFLOPS
AMF-CSR	3.23	3.36
PBC	5.83	5.95
CSR-Scalar	2.30	2.29
CSR5	2.21	2.12

5 结束语

由于稀疏矩阵的不规则性，在SpMV运算过程中，可能出现负载不均衡的现象，根据3.1节中提出的启发式阈值选择算法和基于重排序的行归并算法，提出了一种新的TEB存储格式，该格式根据阈值选择决策树所选的阈值对稀疏矩阵进行重构分解，子块间的相近性最大化，从而满足加速SpMV并行计算负载平衡的要求。在存储过程中，只存储非零元的相关索引，不会造成由零元素引起计算资源的浪费。并且通过实验验证了TEB存储格式的有效性，预处理时间在可以接受的范围内，在SpMV运行时间和浮点运算性能方面，TEB存储格式较于其他格式有较好的效果提升，其中，TEB时间性能较于AMF-CSR、PBC、CSR-Scalar和CSR5平均可提升3.23、5.83、2.33和2.21倍。TEB的浮点运算性能较于AMF-CSR、PBC、CSR-Scalar和CSR5平均可提高3.36、5.95、2.29和2.13倍。在启发式阈值选择算法中，阈值精度为千分位，提高阈值精度会使子块的相似性更高，因此，将在未来工作中结合机器学习算法优化存储格式，来获得更高的性能。

参考文献：

[1] ATEZCAN E, TORUN T, KOSAR F, et al. Mixed and multi-precision SpMV for GPUs with row-wise precision selection[C]//Proceedings of the 2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing, Bordeaux, Nov 2-5, 2022. Piscataway: IEEE, 2022: 31-40.

[2] SUN H Y, GAINARU A, SHANTHARAM M, et al. Selective protection for sparse iterative solvers to reduce the resilience overhead[C]//Proceedings of the 2020 IEEE 32nd International Symposium on Computer Architecture and High Performance Computing, Porto, Sep 9-11, 2020. Piscataway: IEEE, 2020: 141-148.

[3] 李秉政，黄高阳，许瑾晨. 面向申威众核处理器的LZMA并行算法设计与优化[J]. 计算机科学与探索, 2020, 14(9): 1501-1509.
LI B Z, HUANG G Y, XU J C. Design and optimization of parallel LZMA for many-core sunway processor[J]. Journal of Frontiers of Computer Science and Technology, 2020, 14(9): 1501-1509.

[4] YANG M L, DU Y L, SHENG X Q. Solving electromagnetic scattering problems with over 10 billion unknowns with the parallel MLFMA[C]//Proceedings of the 2019 Photonics & Electromagnetics Research Symposium-Fall, Xiamen, Dec 17-20, 2019. Piscataway: IEEE, 2019: 355-360.

[5] LIU J. Accuracy controllable SpMV optimization on GPU [C]//Proceedings of the 2022 4th International Conference on Artificial Intelligence and Computer Science, Beijing, Jul 30-31,2022. Bristol: IOP Publishing, 2022.

[6] AHMED M, USMAN S, SHAH N A, et al. AAQAL: a machine learning-based tool for performance optimization of parallel SPMV computations using block CSR[J]. Applied Sciences, 2022, 12(14): 7073.

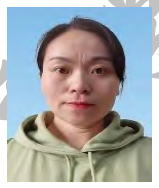
[7] ISOTTON G, FRIGO M, SPIEZIA N, et al. Chronos: a gen-

eral purpose classical AMG solver for high performance computing[J]. SIAM Journal on Scientific Computing, 2021, 43(5): 335-357.
[8] 肖汉,孙陆鹏,李彩林,等.面向GPU的直方图统计图像增强并行算法[J].计算机科学与探索,2022, 16(10): 2273-2285.
XIAO H, SUN L P, LI C L, et al. GPU-oriented parallel algorithm for histogram statistical image enhancement[J]. Journal of Frontiers of Computer Science and Technology, 2022, 16(10): 2273-2285.
[9] CHEN Y D, XIAO G Q, WU F, et al. tpSpMV: a two-phase large-scale sparse matrix-vector multiplication kernel for many-core architectures[J]. Information Sciences, 2020, 523: 279-295.
[10] NAMASHIVAVAM N, MEHTA S, YEW P C. Variable-sized blocks for locality-aware SpMV[C]//Proceedings of the 2021 IEEE/ACM International Symposium on Code Generation and Optimization, Seoul, Feb 27-Mar 3, 2021. Piscataway: IEEE, 2021: 211-221.
[11] BIAN H D, HUANG J Q, LIU L B, et al. ALBUS: a method for efficiently processing SpMV using SIMD and load balancing[J]. Future Generation Computer Systems, 2021, 116: 371-392.
[12] LIU W F, VINTER B. CSR5: an efficient storage format for cross-platform sparse matrix-vector multiplication[C]//Proceedings of the 29th ACM on International Conference on Supercomputing, California, Jun 30-31, 2015. New York: ACM, 2015: 339-350.
[13] ZHANG Y F, YANG W D, LI K L, et al. Performance analysis and optimization for SpMV based on aligned storage formats on an ARM processor[J]. Journal of Parallel and Distributed Computing, 2021, 158: 126-137.
[14] YESIL S, HEIDARSHENS A, MORRISON A, et al. Speeding up SpMV for power-law graph analytics by enhancing locality & vectorization[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, Georgia, Nov 9-19, 2020. Piscataway: IEEE, 2020: 1-15.
[15] CUI H Y, WANG N B, WANG Y H, et al. An effective SPMV based on block strategy and hybrid compression on GPU [J]. The Journal of Supercomputing, 2022, 78(5): 6318-6339.
[16] LI Y S, XIE P Z, CHEN X H, et al. VBSF: a new storage format for SIMD sparse matrix-vector multiplication on modern processors[J]. The Journal of Supercomputing, 2020, 76(3): 2063-2081.
[17] BIAN H D, HUANG J Q, DONG R T, et al. A simple and efficient storage format for SIMD-accelerated SpMV[J]. Cluster Computing, 2021, 24(4): 3431-3448.
[18] GAO J H, JI W X, LIU J, et al. AMF-CSR: adaptive multi-row folding of CSR for SpMV on GPU[C]//Proceedings of the 2021 IEEE 27th International Conference on Parallel and Distributed Systems, Beijing, Dec 14-16, 2021. Piscataway: IEEE, 2021: 418-425.
[19] YANG W D, LI K L, LI K Q. A parallel computing method using blocked format with optimal partitioning for SpMV on GPU[J]. Journal of Computer and System Sciences, 2018, 92: 152-170.
[20] BARRIENTOS E C, INDALECIO G, LOUREIRO A G. Improving performance of iterative solvers with the AXC format using the Intel Xeon Phi[J]. The Journal of Supercomputing, 2018, 74(6): 2823-2840.
[21] BELL N, GARLAND M. Implementing sparse matrix-vector multiplication on throughput-oriented processors[C]//Proceedings of the International Conference for High Performance Computing, Networking, Portland, Nov 14-20, 2009. New York: ACM, 2009: 1-11.
[22] TALATI N, MAY K, BEHROOZI A, et al. Prodigy: improving the memory latency of data-indirect irregular workloads using hardware-software co-design[C]//Proceedings of the 2021 IEEE International Symposium on High-Performance Computer Architecture, Seoul, Feb 27-Mar 3, 2021. Piscataway: IEEE, 2021: 654-667.

王宇华(1977—),男,黑龙江哈尔滨人,博士,副教授,硕士生导师,CCF高级会员,主要研究方向为并行计算、人工智能等。
WANG Yuhua, born in 1977, Ph.D., associate professor, M.S. supervisor, CCF senior member. His research interests include parallel computing, artificial intelligence, etc.

张宇琪(1998—),女,山西太原人,硕士研究生,主要研究方向为高性能计算、并行计算。
ZHANG Yuqi, born in 1998, M.S. candidate. Her research interests include high performance computing and parallel computing.

何俊飞(1998—),男,河南周口人,硕士研究生,主要研究方向为高性能计算、并行计算等。
HE Junfei, born in 1998, M.S. candidate. His research interests include high performance computing, parallel computing, etc.



徐悦竹(1977—),女,博士,讲师,主要研究方向为高性能计算与并行计算、图像识别、语音情感识别、工业大数据等。
XU Yuezhu, born in 1977, Ph.D., lecturer. Her research interests include high performance computing and parallel computing, image recognition, speech emotion recognition, industrial big data, etc.

崔环宇(1988—),男,黑龙江大庆人,博士研究生,主要研究方向为高性能计算、并行计算等。
CUI Huanyu, born in 1988, Ph.D. candidate. His research interests include high performance computing, parallel computing, etc.