面向SW26010-Pro众核处理器的新型矩阵存储格式及稀疏矩阵向量乘(SpMV)算法研究

王 萃[1),2)] 刘芳芳[1),2)] 马文静[1),2)] 赵玉文[1),2)] 胡力娟[1),2)]

[1)](中国科学院软件研究所并行软件与计算科学实验室 北京 100190)

[2)](中国科学院软件研究所基础软件与系统重点实验室 北京 100190)

摘 要 稀疏矩阵向量乘(Sparse Matrix-Vector Multiplication, SpMV)是高性能计算、人工智能大模型领域中的关键操作,其性能通常对应用程序整体性能的提升具有重要影响。高效的稀疏矩阵存储格式是影响SpMV性能的重要因素,然而,现有的稀疏矩阵存储格式主要通过压缩零元素以减少访存,未充分利用非零元素的数值规律,因此仍有进一步压缩和优化的空间。本文通过对压缩稀疏行(Compressed Sparse Row, CSR)存储格式中非零元数组内的重复元素进行进一步的压缩,提出了一种新型的稀疏矩阵存储格式(Further Compressed Sparse Row, FCSR),并设计了从CSR到FCSR格式转换的异构并行算法,以尽量减少格式转换带来的开销。同时,本文面向SW26010-Pro众核处理器,设计了基于FCSR存储格式的SpMV异构并行算法,对SpMV进行了细粒度的任务划分和并行优化设计,探究了五种向量x的间接访存方式,并通过双缓冲技术对算法进行了优化。最后,本文选用SuiteSparse矩阵集中的稀疏矩阵进行了测试,实验结果表明,本文提出的基于FCSR存储格式的异构众核SpMV算法相较于主核版SpMV算法具有明显的性能提升,最高加速比达到43.11,平均加速比为7.56,测试矩阵最高带宽利用率达到了91.13%,平均带宽利用率为26.27%。另外,本文对基于FCSR存储格式和CSR存储格式的SpMV算法性能进行了比较,在两者均得到充分优化的前提下,基于FCSR存储格式的SpMV算法相较于基于CSR存储格式的SpMV算法性能的平均加速比达到1.19。

关键词 稀疏矩阵向量乘;SW26010-Pro众核处理器;新型矩阵存储格式;并行优化;双缓冲技术

中图法分类号 TP301 **DOI号** 10.11897/SP.J.1016.2025.01290

Research on a New Matrix Storage Format and SpMV Algorithm for the SW26010-Pro Many-Core Processor

WANG Cui[1),2)] LIU Fang-Fang[1),2)] MA Wen-Jing[1),2)] ZHAO Yu-Wen[1),2)] HU Li-Juan[1),2)]

[1)](Laboratory of Parallel Software and Computational Science, Institute of Software, Chinese Academy of Sciences, Beijing 100190)

[2)](Key Laboratory of System Software, Institute of Software, Chinese Academy of Sciences, Beijing 100190)

Abstract Sparse Matrix-Vector Multiplication (SpMV) is a key operation in the fields of high-performance computing and artificial intelligence large models, and its performance usually has a significant impact on the improvement of the overall performance of the application. As the scale of the problem continues to increase and the hardware structure continues to evolve, the optimization of SpMV faces challenges such as bandwidth limitation and load imbalance, which

收稿日期:2024-10-24;在线发布日期:2025-03-20。本课题得到国家重点研发计划(2023YFB3001703)、中国科学院软件研究所自主部署科研类项目(ISCAS-JCMS-202304)资助。**王 萃**(通信作者),硕士,工程师,中国计算机学会(CCF)会员,主要研究领域为高性能计算、异构并行、稀疏矩阵相关算法等。E-mail: wangcui@iscas.ac.cn。 **刘芳芳**(通信作者),博士,正高级工程师,中国计算机学会(CCF)会员,主要研究领域为高性能计算、异构并行、超级计算机评测软件等。E-mail: fangfang@iscas.ac.cn。 **马文静**,博士,副研究员,中国计算机学会(CCF)会员,主要研究领域为高性能计算、代码生成与优化等。 **赵玉文**,博士,高级工程师,中国计算机学会(CCF)会员,主要研究领域为高性能计算、异构并行、FFT相关算法等。 **胡力娟**,硕士,助理工程师,中国计算机学会(CCF)会员,主要研究领域为高性能计算、异构并行、BLAS库相关算法等。

has become the difficulty and focus of improving the performance of many applications. In order to effectively improve the performance of SpMV, it is necessary to fully tap the potential of the underlying system architecture and improve the utilization of data transmission bandwidth. At the same time, efficient sparse matrix storage format is also one of the key factors affecting SpMV performance. However, the existing storage formats mainly compress zero elements to reduce memory access, but does not fully utilize the numerical regularity of non-zero elements, leaving room for further compression and optimization. This paper further compresses the repeated elements in the non-zero element array in the Compressed Sparse Row (CSR) storage format, proposes a new sparse matrix storage format (Further Compressed Sparse Row, FCSR), and designs a heterogeneous parallel algorithm for converting from CSR to FCSR format to minimize the overhead caused by format conversion. At the same time, this paper designs a heterogeneous parallel SpMV algorithm based on the FCSR storage format for the SW26010-Pro many-core processor. The algorithm involves fine-grained task partitioning and parallel optimization, explores five indirect memory access methods for vector x, and optimizes the algorithm using double buffering techniques. Finally, we select sparse matrices from the SuiteSparse matrix set for testing. Experimental results demonstrate that the proposed heterogeneous many-core SpMV algorithm based on the FCSR storage format outperforms the controller corevision, achieving a maximum speedup of 43.11 and an average speedup of 7.56. The highest bandwidth utilization of the test matrix reached 91.13%, and the average bandwidth utilization was 26.27%. In addition, this paper compares the performance of SpMV algorithms based on FCSR storage format and CSR storage format. On the premise that both are fully optimized, the average speedup ratio of the SpMV algorithm based on the FCSR storage format to the SpMV algorithm based on the CSR storage format reaches 1.19. The FCSR storage format and its format conversion algorithm proposed in this paper are not limited to a specific platform and can be applied to a variety of mainstream multi-core/many-core processors. At the same time, the SpMV heterogeneous parallel algorithm based on the FCSR storage format proposed in this paper can be implemented on similar heterogeneous processors such as SW26010. Among them, the applicability of indirect memory access methods of various vectors on different platforms is different, and it needs to be selected according to the actual situation when applied. SpMV calculations are widely used in natural language processing, deep learning, image processing and other fields. For matrix-vector multiplication calculations with high matrix compression rate, FCSR format can be considered for storage and calculation. On the other hand, as more and more artificial intelligence problems migrate to heterogeneous platforms such as SW26010-Pro, the SpMV optimization solution proposed in this paper has important reference significance for solving such problems.

Keywords sparse matrix-vector multiplication; SW26010-Pro multicore processor; new sparse matrix storage format; parallel optimization; double buffering technology

1 引　言

稀疏矩阵向量乘(SpMV)$\boldsymbol{y}=\boldsymbol{Ax}$在科学[1]与工程领域[2-3]中扮演着至关重要的角色，在大模型训练[4-5]与推理[6]等前沿领域也发挥着关键作用。然而，随着问题规模的不断增大以及硬件结构的持续演进[7-8]，SpMV的优化面临着带宽受限和负载不均衡等挑战，已成为许多应用性能提升的难点和重点。为了有效提升SpMV的性能，必须充分挖掘底层系统架构的潜力，提高数据传输带宽的利用率。同时，高效的稀疏矩阵存储格式也是影响SpMV性能的关键因素之一。

SW26010-Pro是我国自主研发的新一代申威

异构众核处理器，相比于SW26010处理器，其LDM（Local Data Memory）空间和向量化宽度增大，计算核心之间数据交换方式也由寄存器通信转变为RMA（Remote Memory Access）。现有SW26010上的优化方案不足以充分发挥SW26010-Pro的硬件特性[9]，因此针对SW26010-Pro处理器特有的体系结构，重新设计SpMV的内存分配方式及并行优化算法十分必要。另一方面，稀疏矩阵具有极高的稀疏性和不规则性，稀疏矩阵的存储格式成为提升计算性能的关键因素之一。有效的稀疏矩阵存储格式能够节省存储空间，充分利用处理器体系结构的优势，从而优化性能。常用的稀疏矩阵存储格式包括CSR、COO、DIA、ELL等[10]，这些存储格式使用一维或二维数组来存储非零元素及其相关的索引信息，其中，CSR格式存储格式紧凑高效，已成为科学与工程领域中最为广泛应用的格式之一[11]。然而，在科学计算领域，稀疏矩阵通常来源于各种实际问题的数学建模和离散化过程，由于应用问题本身的规律性以及这些离散化方法的特点，稀疏矩阵往往呈现出一定的规律性和重复性。通过充分利用这些规律和重复性，现有稀疏矩阵存储格式中的非零元素部分具有进一步压缩的潜力。

本文对CSR存储格式中的非零元素部分进行了进一步的压缩，提出了一种新型的稀疏矩阵存储格式FCSR。同时，设计了从CSR到FCSR格式转换的异构并行算法，以减少转换过程中的开销，该格式转换算法具有良好的移植性，可以在各种不同的多/众核平台上应用。

同时，本文深入分析了国产SW26010-Pro平台的特性，提出了基于FCSR存储格式的SpMV异构并行算法，对SpMV进行细粒度的任务划分和并行优化设计，探索了多种向量x的间接访存方式，并利用双缓冲技术对算法进行优化。本文提出的基于FCSR存储格式的SpMV异构并行算法可在SW26010等类似异构硬件平台上实现移植应用，其中，多种向量x的间接访存方式在不同平台上的适用性有所不同，移植应用时需根据实际情况选择。

针对人工智能大模型应用场景下具有较高可压缩率的矩阵，可考虑采用FCSR格式进行存储和SpMV计算。同时，随着人工智能算力需求向SW26010-Pro等异构计算平台延伸，本文提出的SpMV优化方案对于解决这类问题也具有重要参考意义。

2 背 景

2.1 国产SW26010- Pro众核处理器

本文研究工作基于国产SW26010-Pro处理器开展。该处理器的单芯片由六个核组构成，如图1所示，每个核组包括一个控制管理核心（主核，MPE）以及一个由64个异构计算核心（从核，CPE）组成的阵列。主核主要负责前处理和控制任务，从核主要承担计算任务。每个核组内配备了16 GB内存，主核和从核阵列共享一个128位的DDR4内存控制器，DDR4数据传输速率为3200 Mbps，核组的理论带宽为51.2 GB/s。单个核组主存的实测访存带宽约为46 GB/s，实测带宽占理论带宽的89.84%。

每个从核本地独占一个私有高速局部存储器（Local Data Memory，LDM），大小为256 KB，该空间为可编程高速缓存的模式，可将32 KB或128 KB配置成硬件自动管理的数据高速缓冲存储器（Cache）。从核具有快速且低延迟的LDM读写能力，如果系统配置了Cache，直接从主存读取的数据会被存储到Cache中，缓存行长度为64 B。从核还支持使用直接内存访问（Direct Memory Access，DMA）的方式在主存和LDM之间快速传输数据，在发起异步DMA通信后，从核可以同时执行其他计算任务。DMA数据传输的粒度大小对传输带宽有显著影响，传输大块连续数据的效率要高于传输小块不连续数据。另外，从核还可以通过远程内存访问（Remote Memory Access，RMA）机制，完成本地从核和远程任一从核之间的数据传输。RMA机制的带宽和延迟，取决于从核之间的相对位置，以及网络拓扑结构，RMA接口同样也支持异步通信。SW26010-Pro处理器还提供了包括远程地址访问（Remote Load access，RLD）从核通信功能，通过远程位置访问数据，RLD操作的硬件开销显著低于数据缓存。

2.2 CSR存储格式与SpMV算法

矩阵存储格式CSR是一种基于行主序的高效矩阵压缩存储格式，通过压缩行的方式来存储矩阵，该格式仅记录非零元素的值及其对应的列索引，从而显著减少了存储空间的使用。CSR格式的存储结构主要包括三部分：非零元值数组（*Val*），列索引数组（*Col*）和行指针数组（*Ptr*），具体如图2所示：

非零元值数组（*Val*）：存储矩阵中非零元素的

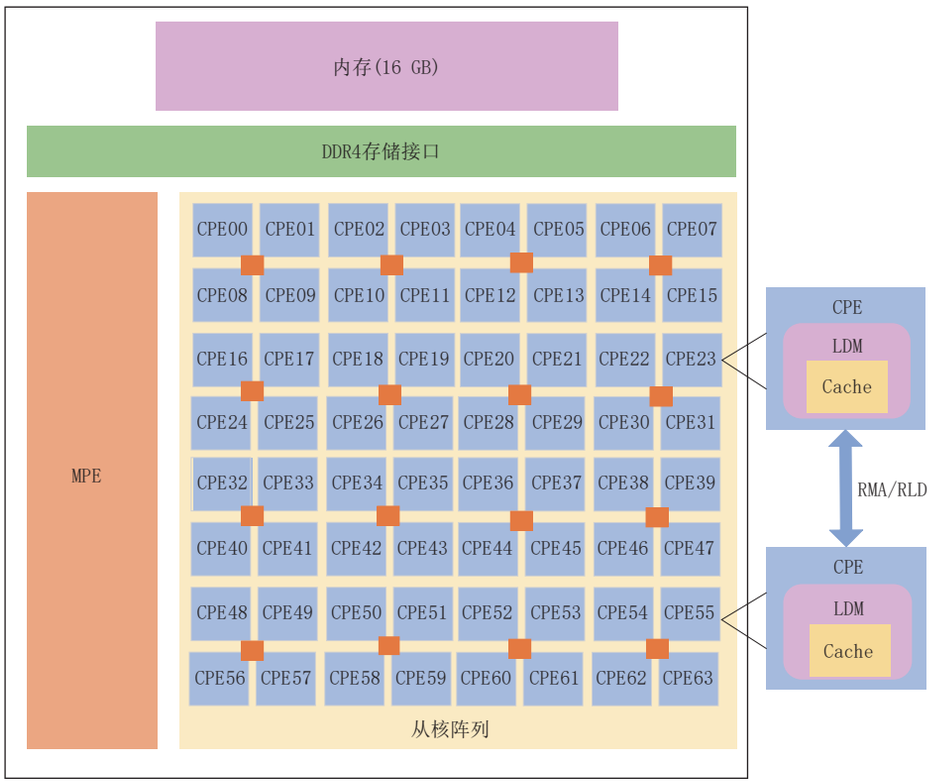

图1　SW26010-Pro处理器架构示意图

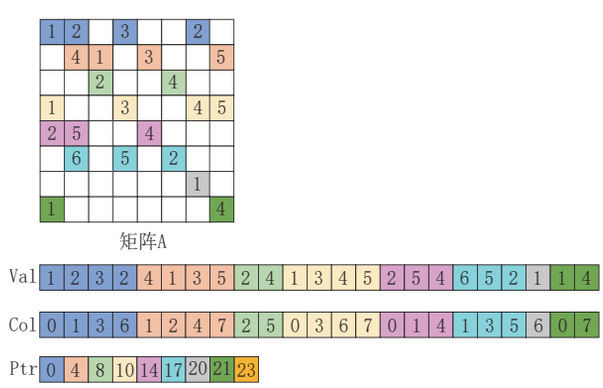

图2　CSR格式示意图

值，按行主序依次排列。

列索引数组(*Col*)：存储每个非零元素对应的列索引，按行主序依次排列。

行指针数组(*Ptr*)：存储每行的起始索引，指示了值数组和列索引数组中每一行的起始位置。

在基于CSR格式的稀疏矩阵向量乘算法中，首先通过行指针数组*Ptr*确定每行中非零元素的数量及起始位置，然后依次读取非零元数组*Val*中的非零元素，并根据列索引数组*Col*获取每个非零元素所需的向量x数组中的对应数值，通过循环逐行完成矩阵和向量元素的乘加操作。

设所计算矩阵的维数为n，非零元总数为nz，则基于CSR的SpMV算法总的内存访问量为：

$$SCSR=(Sval+Scol)\times nz+(Sx+Sy)\times n+Sptr\times(n+1) \tag{1}$$

其中*Sval*、*Scol*、*Sptr*、*Sx*和*Sy*分别为非零元值数组*Val*、列索引数组*Col*、行指针数组*Ptr*、向量$\boldsymbol{x}$以及向量$\boldsymbol{y}$中一个元素占用的字节数。本文矩阵非零元*Val*和向量$\boldsymbol{x}$和$\boldsymbol{y}$均采用双精度浮点数据类型，即占用8个字节，*Ptr*和*Col*为整型数据类型，占用4个字节，则基于CSR的SpMV算法总的内存访问量为：

$$SCSR=12\times nz+20\times n+4 \tag{2}$$

2.3　相关工作

稀疏矩阵向量乘法(SpMV)在科学计算中占据重要地位。每当新型处理器推出时，针对该处理器的SpMV实现和优化工作便会不断涌现。

多核CPU是一种通用处理器，研究人员主要通过多线程并行[12]和向量化[13]来提升CPU上SpMV计算性能。近年来，相关研究人员已经针对CPU上稀疏矩阵存储格式的优化进行了研究。CSR-DU[14]、CSX[15]、VHCC[16]、CVR[17]、CSR2[18]、M-HDC[19]等存储格式先后被提出，这些新的存储格式通过列索引和行指针低字节存储、多种存储格式混合、分块存储等手段对现有的稀疏矩阵存储格式进行改进，提高了内存访问效率，进而提升了多核

CPU上SpMV的性能。

随着GPU加速器的快速发展，SpMV算法在GPU上的优化工作引起了越来越多的关注。GPU上的SpMV优化研究涵盖了向量化技术[20-21]、分块算法[22-23]、自适应调优[24-25]和负载均衡[26-27]等。面向GPU的稀疏矩阵存储格式的优化工作也陆续展开，CSR5[28]、AXC[29]、HYB[30]、TEB[31]、DCSR[32]等新型矩阵存储格式先后被提出，这些新的矩阵存储格式通过向量化、分块、格式融合等方法对传统的稀疏矩阵存储格式进行改进。随着人工智能的发展，一些研究人员结合机器学习模型来训练和分析稀疏矩阵的特征点，然后根据训练结果为不同的矩阵自适应选择存储格式[33]。

随着国产异构众核处理器的发展，面向国产异构众核处理器SpMV算法的研究也陆续展开。Liu等人[34]在SW26010处理器上提出了一种基于CSR存储格式的SpMV异构并行算法，从任务划分、LDM空间划分等方面优化SpMV性能。Li等人[35]在SW26010处理器上针对稀疏矩阵向量乘法，基于线程级和指令级并行层面进行细粒度的并行算法设计和优化实现。Ma等人[36]在SW26010-Pro处理器上实现了基于BCSR(Block Compressed Sparse Row)存储格式的SpMV算法，通过动态任务调度实现负载均衡。Wang等人[37]面向SW26010-Pro处理器，提出了一种SPM聚合策略和向量自适应映射算法，通过两级数据分区方案来实现计算负载平衡。

高性能计算平台上的SpMV相关研究表明，不论是CPU等同构计算平台，还是GPU、申威等异构计算平台，SpMV的优化主要集中在合理的任务划分、向量化技术以及稀疏矩阵存储格式的优化等方面。然而，SW26010-Pro处理器所引入的许多新机制尚未得到充分发掘，其在SpMV性能方面的潜力仍有待进一步探索。同时，现有的稀疏矩阵格式大多是针对矩阵中零元素部分的压缩，而忽略了非零元素部分的重复性，仍存在进一步压缩的潜力。

本文主要从非零元素的重复性入手，提出了一种新型稀疏矩阵存储格式FCSR，并介绍了一种从CSR到FCSR格式转换的异构并行算法。同时，本文还提出了基于FCSR存储格式的SpMV异构并行算法，对SpMV进行细粒度的任务划分和并行优化设计。

3 新型稀疏矩阵存储格式FCSR

第3.1节对稀疏矩阵非零元素的可压缩率进行了分析，第3.2节介绍了FCSR存储格式的具体实现方式，第3.3节详细介绍了从CSR到FCSR格式转换的异构并行算法。

3.1 稀疏矩阵非零元规律

在科学计算领域，稀疏矩阵通常来源于各种实际问题的数学建模和离散化过程，由于应用问题本身的规律性以及这些离散化方法的特点，稀疏矩阵往往呈现出一定的规律性和重复性。基于这些规律和重复性，稀疏矩阵存储格式中的非零元素部分存在进一步压缩的潜力。本文以CSR存储格式为基础，探究其非零元素的可压缩性。

设稀疏矩阵全部非零元数为nz，压缩后不重复的非零元数nnz，定义稀疏矩阵可压缩率α为

$$\alpha=\frac{nz-nnz}{nz} \tag{3}$$

稀疏矩阵的可压缩率越高，表明稀疏矩阵非零元素中重复的元素占比越高。我们对SuiteSparse矩阵集中的上千个稀疏矩阵的可压缩率进行了统计，结果如图3所示。稀疏矩阵的可压缩率超过99%的矩阵占总数的10%，可压缩率超过90%的矩阵占总数的22%，可压缩率超过50%的矩阵则高达78%。

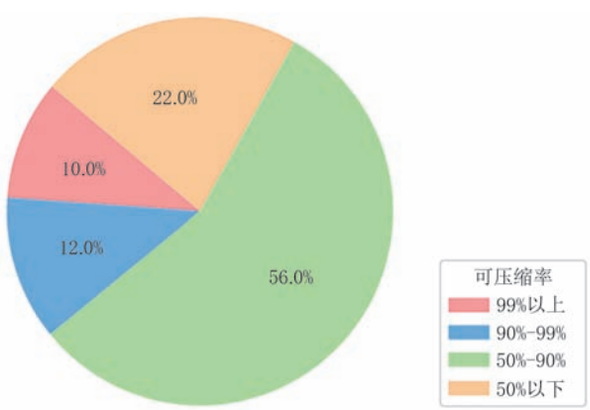

图3 稀疏矩阵可压缩率占比情况

统计结果表明，绝大部分稀疏矩阵包含大量的重复非零元素，具有较高的可压缩率。通过压缩这些重复部分，可以显著减少非零元素的存储需求和数据传输量。利用稀疏矩阵中非零元素高度重复的特性，探索新型存储格式对于提升SpMV计算效率具有重要意义。

3.2 新型稀疏矩阵存储格式FCSR

本文以CSR存储格式为基础，对非零元数组的存储方式进行改进，提出了一种新型的稀疏矩阵存储格式FCSR。该存储格式引入了*Acompress*和

*Aindex*两个数组来替代CSR存储格式中的非零元数组*Val*，以更高效地存储非零元素。图4为本文提出的新型稀疏矩阵存储格式FCSR的存储示意图，*Col*和*Ptr*数组与CSR对应部分相同，新增两个数组的具体情况如下：

压缩后的非零元数组(*Acompress*)：存储矩阵中不重复的非零元素的值。

非零元索引数组(*Aindex*)：存储原非零元数组*Val*中每个元素在压缩后非零元数组*Acompress*中的相对位置，按行主序依次排列。

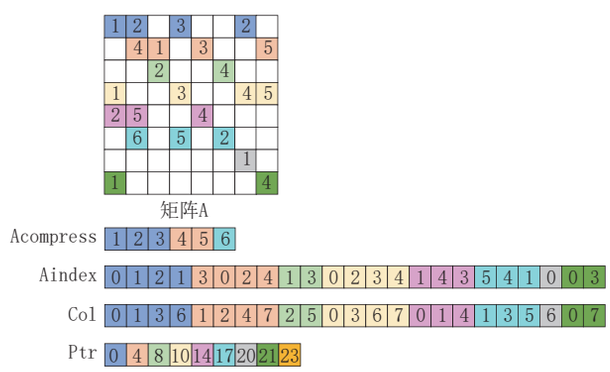

图4　新型稀疏矩阵存储格式FCSR示意图

*Acompress*数组中的元素采用双精度浮点类型存储，占8个字节，*Aindex*数组中采用整数类型存储占4个字节，当压缩后的矩阵非零元小于65 536时，*Aindex*数组可采用无符号短整型存储，占2个字节。当稀疏矩阵具有较高的可压缩率时，FCSR存储格式中*Acompress*和*Aindex*两个数组的存储空间将明显小于原CSR存储格式中*Val*数组的存储空间。

3.3　从CSR到FCSR格式转换的异构并行算法

从CSR到FCSR的格式转换过程中，不同数组之间以及数组内部存在着依赖关系，如果采用串行处理方式，格式转换的开销较大。本文设计了SW26010-Pro异构众核处理器上从CSR到FCSR格式转换的并行算法，其中，主核负责任务调度，从核负责完成实际的格式转换工作，通过多个从核的同时协作，可显著提升格式转换的效率。

设矩阵A的总行数为n行，我们将n行的矩阵格式转换任务分配给np个从核并行完成，每个从核分别负责完成m行的格式转换，其中$m=n/np$。由于从核LDM存储空间有限，每个从核的格式转换任务需通过多次循环完成，设每个从核单次可完成格式转换的行数为$srow$，循环k次完成m行的格式转换工作，其中$k=m/srow$。为了避免格式转化过程中不同从核之间的数据依赖，每个从核独立完成其负责行的非零元从CSR到FCSR的格式转换工作，从核之间无需进行任何交互。

图5为SW26010-Pro异构众核处理器上从CSR到FCSR格式转换的示意图。每个从核独立构建各自的*Acompress*和*Aindex*数组，数组长度为m行矩阵非零元素长度的最大值。每次循环首先通过DMA读取CSR存储格式中$srow$行指针数组*Ptr*和非零元数组*Val*，然后遍历每一行的非零元，根据从核中非零元的重复情况更新*Acompress*和*Aindex*数组。*Count*数组用于依次记录每个从核内部非零元数组*Acompress*中实际存储非零元的个数num。

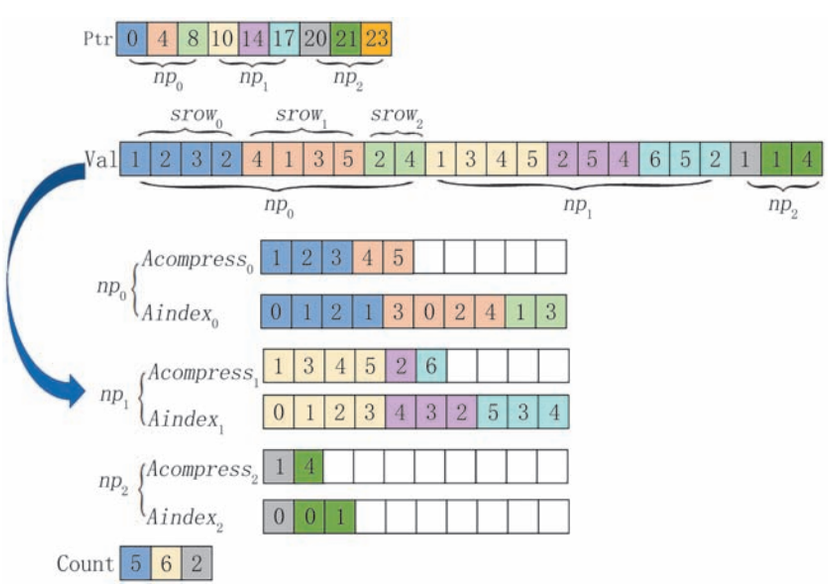

图5　SW26010-Pro处理器上从CSR到FCSR格式转换示意图

算法1为SW26010-Pro异构众核处理器上从CSR到FCSR存储格式转换的具体实现。首先创建一个关联容器，用于存储CSR格式中的非零元素及FCSR格式中对应的索引值。同时创建*pair*容器，该容器包含两个元素，第一个元素为前面创建的关联容器，用于存储插入的键值对，第二个元素是布尔值，用于表示插入操作是否成功。

算法1. 从CSR到FCSR格式转换的异构并行算法

1： 初始化$num=0$，定义$MYID$为从核号
2： 创建$map(double,\text{int})$关联容器
3： 创建$pair<map::\text{int}erator,bool>$容器
4： **for** $srow$ from $srow_0$ to $srow_k$ **do**
5： 读取$srow$行的行指针数组和非零元数组
6： 定义计数器$count=0$
7： **for** i from 0 to $Ptr[srow]-Ptr[0]$ **do**
8： **if** insert $map(Val[i],num)$success **then**
9： $Acompress[count]=Val[i]$
10： $num++$
11： $count++$
12： $Aindex[i]=num$
13： **else**
14： $Aindex[i]=num+1$
15： **end if**
16： **end for**
17： 存储$Acompress$和$Aindex$数组
18： **end for**
19： $Count[MYID]=num$

每个从核对所负责矩阵行的非零元进行格式转换，设参与格式转换的从核数量为np，矩阵总行数为n，CSR中原非零元总数为nz，格式转换后所有从核的非零元数之和为nnz，$Sacom$和$Sain$分别为压缩后非零元数组$Acompress$和非零元索引$Aindex$数组元素占用的字节数，Sc为$Count$数组元素占用字节数，则基于FCSR存储格式的SpMV总访存量为

$$SFCSR=Sacom\times nnz+(Sain+Scol)\times nz+(S\boldsymbol{x}+S\boldsymbol{y})\times n+Sptr\times(n+1)+Sc\times np \quad (4)$$

本文选择64个从核同时进行格式转换，即$np=64$，$Count$数组元素采用整数类型，即$Sc=4$，矩阵和向量都选择双精度浮点类型存储，即$Sacom=8$，若$Aindex$数组元素采用整数类型存储，$Sain=4$，则总访存量为

$$SFCSR=8\times nnz+8\times nz+20\times n+260 \quad (5)$$

当格式转换后的矩阵非零元数小于65 536时，$Aindex$数组元素可采用无符号短整数类型存储，即$Sain=2$，此时总访存量为

$$SFCSR=8\times nnz+6\times nz+20\times n+260 \quad (6)$$

公式(2)为基于CSR存储格式的SpMV总访存量，通过公式(2)和公式(5)，可推算出基于CSR存储格式的SpMV总访存量$SCSR$和基于FCSR存储格式的SpMV总访存量$SFCSR$之间的差值为

$$SCSR-SFCSR=4\times nz-8\times nnz-256 \quad (7)$$

根据公式(7)可以推出，当满足$nnz<nz/2-32$时，基于FCSR存储格式的SpMV相较于基于CSR存储格式的SpMV具有更低的访存量。结合公式(3)推导可得，当矩阵压缩率α满足$\alpha>0.5+32/nz$时，基于FCSR存储格式的SpMV算法具有访存优势。

虽然初始的格式转换会造成一定的开销，但在实际应用中，由于SpMV计算通常需要进行大量迭代，而格式转换仅需要在计算之前执行一次，因此格式转换的成本相对较低。

4 基于FCSR存储格式的SpMV异构并行算法

本文设计了SW26010-Pro众核处理器上基于FCSR存储格式的SpMV并行算法。其中，第4.1节介绍了SpMV算法在SW26010-Pro众核处理器上的任务分配方式，第4.2节探讨了LDM空间的划分策略，第4.3节提出了五种向量$\boldsymbol{x}$的间接访存方式，第4.4节介绍了基于FCSR存储格式的SpMV算法中双缓冲技术的应用。

4.1 任务分配

在进行稀疏矩阵-向量乘法计算时，首要任务是合理分配稀疏矩阵和向量的数据到各个从核上，以充分利用计算资源和访存带宽。图6为SpMV算法在SW26010-Pro众核处理器上的任务分配方式。

设矩阵$\boldsymbol{A}$的总行数为n行，n行SpMV任务分到np个从核上完成，每个从核负责完成m行的矩阵$\boldsymbol{A}$与非零元对应的向量$\boldsymbol{x}$的乘法运算，并将结果存放到结果向量$\boldsymbol{y}$中对应的位置，其中$m=n/np$。由于每个从核的LDM空间有限，当稀疏矩阵的维度或非零元素数量较大时，每个从核需要通过多次循环才能完成所分配行的SpMV任务，设每个从核单次计算可完成的计算的行数为$srow$行，从核内部循环k次完成m行的格式转换工作，其中$k=m/srow$。

4.2 LDM空间划分

SW26010-Pro众核处理器中每个从核拥有

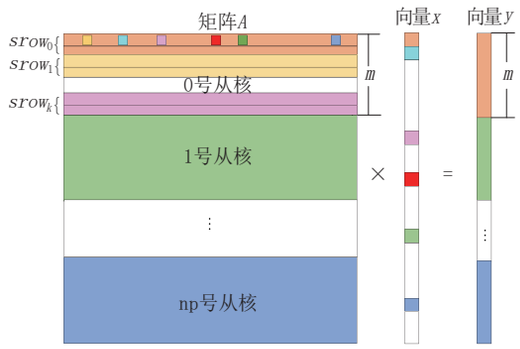

图6　SW26010-Pro处理器上SpMV的任务分配

256 KB的LDM空间，其中32 KB或128 KB可配置成高速缓存区，从核访问LDM中的数据只需要几个时钟周期，而访问主存则需要100多个时钟周期，充分利用从核LDM空间对于并行算法的设计至关重要。

本文将采用双缓冲技术进行SpMV计算，按照4.1节中的任务分配方法，在每个从核上进行一次SpMV计算时，需要在LDM中存储$2\times srow$行的矩阵信息以及m行的向量信息，其中m为每个从核负责完成SpMV的计算行数。设每个矩阵行中非零元素的最大数量为$nz_{\max}$，则LDM空间分配情况如表1所示：

表1　LDM空间分配情况

矩阵或向量	字节数	元素总个数	需分配的LDM空间
Acompress	8	$m\times nz_{max}$	$8\times m\times nz_{max}$
Aindex	4或2	$2\times srow\times nz_{max}$	$8\times srow\times nz_{max}$/ $4\times srow\times nz_{max}$
Ptr	4	$2\times(srow+1)$	$8\times(srow+1)$
Col	4	$2\times srow\times nz_{max}$	$8\times srow\times nz_{max}$
x	8	m	$8\times m$
y	8	m	$8\times m$

若*Aindex*数组元素采用整数类型，占用字节数为4，代入其他矩阵和向量占用的字节数，一个从核单次SpMV计算需分配的LDM空间*SCPE*为

$$SCPE=(16\times nz_{\max}+8)\times srow+(16+8\times nz_{\max})\times m+8 \tag{8}$$

若*Aindex*数组元素采用无符号短整数类型，占用字节数为2，代入其他矩阵和向量占用的字节数，则一个从核单次SpMV计算需分配的LDM空间*SCPE*为

$$SCPE=(12\times nz_{\max}+8)\times srow+(16+8\times nz_{\max})\times m+8 \tag{9}$$

如果不设置高速缓存区时，*SCPE*的最大值为256 KB，如果设置32 KB高速缓存区，*SCPE*的最大值为224KB。当确定矩阵后，便可确定公式(8)和公式(9)中的参数m和$nz_{\max}$，从而计算出从核单次可计算的最大行数$srow$。然后，根据表1的最后一列，可以计算出矩阵各个部分和向量在LDM中的最大申请空间。

4.3　向量x的间接访存方式

在稀疏矩阵向量乘法中，向量x的访存行为是间接且高度不规则的，这是影响性能的关键因素之一。SW26010-Pro处理器引入了一些新的硬件特性，利用这些特性，探索不同的向量x间接存取方式对于提升SpMV性能具有重要意义。

本文在之前基于SW26010处理器提出的向量x访存优化工作的基础上[34]，对向量x的访存方式进行了改进，提出了在SW26010-Pro处理器上五种不同的向量x间接访存方式。图7为这五种向量x间接访存方式的示意图，介绍具体如下：

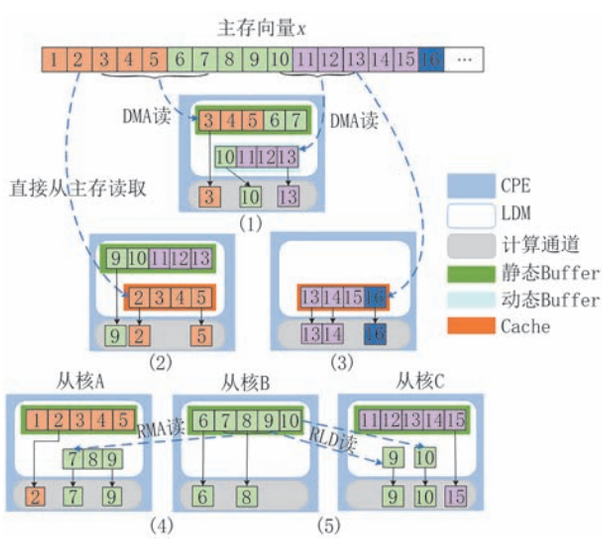

图7　向量x的间接访存方式示意图

(1) DMA+Dynamic Buffer：无需设置高速缓存区，每个从核通过DMA将部分向量x预取到静态Buffer，其余部分通过动态Buffer获取。预取到静态Buffer的起始位置为当前从核计算所需的第一个向量x的位置，预取的向量长度需根据实际矩阵调整，具体实现如图7中(1)所示。在矩阵向量乘计算过程中，先检查静态Buffer中是否包含所需向量x，如果没有，则以当前从核所需向量x的位置为起始位置，读取128个向量元素到动态Buffer，动态Buffer的向量在后续计算中可被更新或重复利用。

(2) DMA+Cache Read：设置32 KB的高速缓

存区，每个从核通过DMA预取部分向量x存到静态Buffer，其余向量x通过直接访问主存获得，预取方式与前述(1)相同，具体实现如图7中(2)所示。在计算过程中，如果所需的向量x不在静态Buffer中，则根据列索引从主存中的相应位置读取。

(3) Cache Read：设置32 KB的高速缓存区，所有的x均通过直接访问主存的方式从主存中获取，具体实现如图7中(3)所示。由于缓存行的存在，直接访问主存可以一次获取8个双精度浮点类型的向量元素，从主存中直接读取的向量元素会被存储在Cache中，在后续行的计算中可反复重用。

(4) DMA+RMA：无需设置高速缓存区，每个从核通过DMA将部分向量x预取到静态Buffer，其余通过RMA从其他从核中读取。预取到静态Buffer的起始位置固定，预取的向量长度需确保所有从核拥有完整的向量x。具体实现如图7中(4)所示，从核A通过RMA远程读取从核B中的向量段，读取起始位置为当前从核计算所需向量x的位置，读取的长度设为128个向量元素。通过RMA读取的向量在后续计算中可被重复利用。

(5) DMA+RLD：无需设置高速缓存区，每个从核通过DMA将预取部分x到静态Buffer，其余通过RLD从其他从核中获取，预取方式与前述(4)相同。具体实现如图7中(5)所示，从核C通过RLD远程访问从核B中向量x的地址，进而获取所需的向量x的值。

这五种方法分别利用了SW26010-Pro处理器的多种硬件特性。后续测试将对每种读取方式的适用性和性能进行比较和评估。

4.4 双缓冲技术

该处理器的从核支持DMA访存与计算的重叠执行。为了验证此重叠机制的有效性，本文在LDM上设置了*Aindex*、*Col*和*Ptr*的双缓冲区。通过计算访存重叠掩盖访存开销。算法2为基于FCSR存储格式的异构众核SpMV算法，算法采用了双缓冲技术。

算法2 基于FCSR存储格式的异构众核SpMV算法

```
1:  读取所需行的Acompress和srow_0行的Ptr
2:  读取srow_0行的Aindex和Col
3:  DMA wait
4:  for srow_i from srow_1 to srow_n do
5:    读取srow_i行的Ptr
6:    读取srow_i行Aindex和Col
7:    初始化jj=0
8:    for i from 0 to srow do
9:      y[i]←0
10:     for j form Ptr[i] to Ptr[i+1] do
11:       ai=Acompress[Aindex[jj]]
12:       y[i]←y[i]+ai×x[Col[jj]]
13:       jj=jj+1
14:     end for
15:   end for
16:   DMA wait
17:   存放srow行的结果向量
18:   end for
```

图8为基于FCSR格式SpMV算法的时序图，上面为单缓冲的时序图，下面为双缓冲的时序图，由于矩阵A和向量x均存在间接访存，其内存访问不连续，访存时间较长，使得算法2中8-15行的运行时间较长。通过计算访存重叠，可以有效地减少整体运算时间。

5 实验结果及分析

我们在SW26010-Pro处理器的一个核组上进行了测试，评估了本文优化的不同版本的SpMV并行算法在该平台上的性能。首先，测试了基于五种向量x间接访存方式的SpMV的性能；然后评估了双缓冲技术和新型矩阵存储格式FCSR所带来的优化效果；最后，我们将本文优化后的基于FCSR存储格式的SpMV与原主核版本的SpMV进行了性能对比，测试了带宽利用率，评估了本文提出的SpMV算法的整体效果。实验中使用的矩阵和向量的数据类型为双精度浮点数。

5.1 测试矩阵

我们首先从SuiteSparse矩阵集中选择了14个具有代表性的稀疏矩阵，分别评估了向量x间接访存方式、双缓冲技术以及FCSR存储格式为SpMV算法带来的性能提升。在表2中列出了这14个测试矩阵的基本信息，包括矩阵的维数、非零元素数量以及每行平均非零元素数量。此外，我们还从SuiteSparse矩阵集中另外选择了508个规模较大的实数矩阵，这些矩阵的非零元素数量分布在10^3到10^7范围内。基于这些矩阵，我们对本文提出的基于FCSR存储格式的SpMV算法进行了整体性能评估。

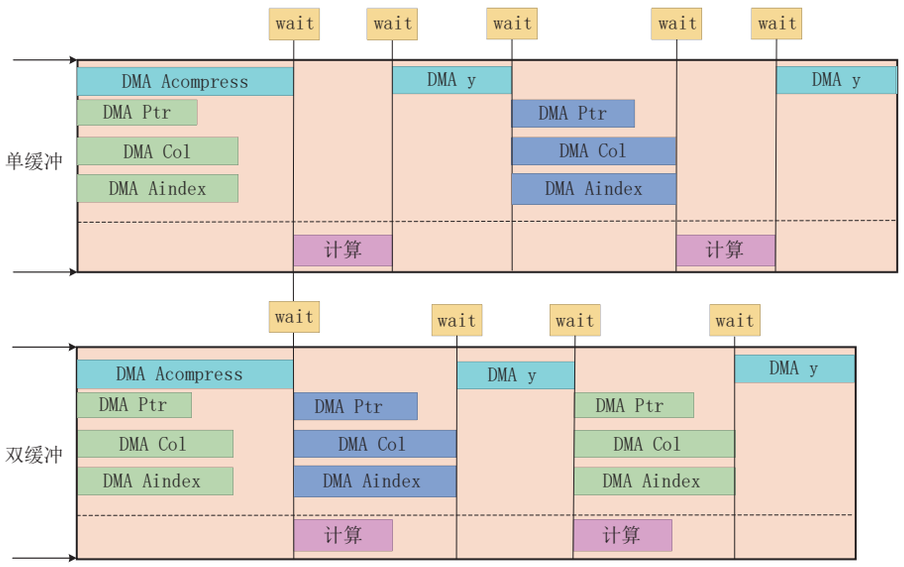

图8　基于FCSR格式SpMV算法的时序图

表2　测试矩阵信息表

编号	矩阵名字	矩阵维数	非零元数	平均每行非零元数
1	fv2	9801	87 025	8.88
2	torsion1	40 000	197 608	4.94
3	mark3jac120sc	54 929	342 475	6.23
4	bcsstk16	4884	290 378	59.45
5	bratu3d	27 792	173 796	6.25
6	qa8fm	66 127	1 660 579	25.11
7	bcsstk18	11 948	149 090	12.48
8	t2d_q4	9801	87 025	8.88
9	bloweybq	10 001	69 991	7.00
10	s3dkq4m2	90 449	4 820 891	53.30
11	hvdc2	189 860	1 347 273	7.09
12	xenon2	157 464	3 866 688	24.56
13	qa8fk	66 127	1 660 579	25.11
14	venkat01	62 424	1 717 792	27.52

5.2　不同向量x间接访存方式效果评估

我们对基于第4.3节中提出的五种向量x访存方式的SpMV运行时间进行了测试。矩阵首先转换为FCSR存储格式，然后统计了10个稀疏矩阵基于不同向量x访存方式的SpMV运行1000次的时间，测试结果如图9所示。由于不同矩阵的非零元素分布存在差异，不同向量x访存方式对各个矩阵的SpMV性能影响也不同。

从图9可以看出，五种向量x访存方式中，基于“Cache Read”方式获取向量x时SpMV的性能最佳，基于“DMA+Cache Read”方式获取向量x时SpMV性能次优。这两种方法直接从主存中读取的向量x会被存储在高速缓存Cache中。由于处理器

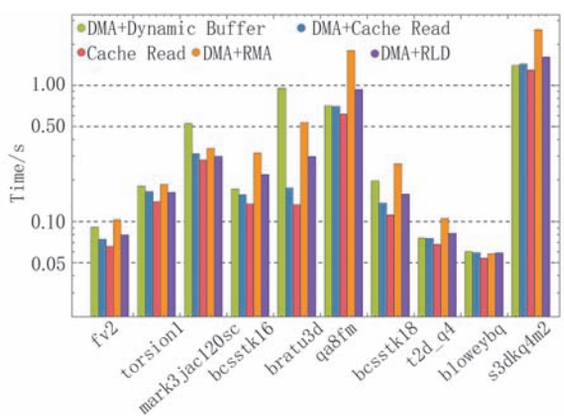

图9　基于五种向量x访存方式的SpMV的运行时间的对比

拥有较大的Cache空间，同时支持缓存行，一次访问主存可读取8个向量x，向量x重用率高，使得这两种方式展现出较好的性能优势。

虽然基于RLD操作的硬件开销较低，但通过“DMA+RLD”方式获取的向量x无法重复利用，当稀疏矩阵非零元数较多或分布较密集时，基于“DMA+RLD”方式的SpMV性能表现较差，该方式更适合非零元数较少且较稀疏的矩阵。对于非零元素数量较多的矩阵，如“qa8fm”和“s3dkq4m2”，基于“DMA+Dynamic Buffer”方式获取向量x的SpMV展现出较好的性能优势。而在使用RMA传输时，同一从核与不同从核之间的数据交换会存在等待时间，因此基于“DMA+RMA”方式获取向量x时SpMV性能表现较差。

在本文实验中，通过athread_spawn()接口启动从核时，每次启动默认会刷新Cache，确保Cache清空，从而使得Cache命中率不会受到多次测试的影

响。因此，“Cache Read”方式被视为最佳选择，不受测试方法的影响。在后续的SpMV测试中，我们将采用全部从主存中直接读取的“Cache Read”方式获取向量x。

5.3 双缓冲技术效果评估

为探究双缓冲技术对稀疏矩阵SpMV计算性能的影响，我们选取了10个稀疏矩阵，并将它们统一转换为FCSR存储格式，然后分别在使用和未使用双缓冲技术的条件下，进行1000次SpMV运算。测试结果如图10所示。引入双缓冲技术后，SpMV的运行时间有所减少，平均加速比为1.06，最高加速比为1.12。因此，双缓冲技术可有效提高SpMV的执行效率，后续的测试将采用使用双缓冲技术的SpMV进行测试。

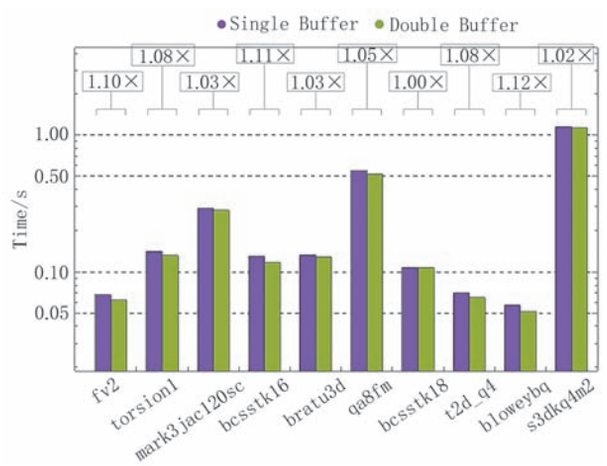

图10 引入双缓冲技术前后的运行时间对比

5.4 不同版本的FCSR存储格式效果评估

本文的3.2节介绍了FCSR存储格式中非零元索引数组*Aindex*中的元素可采用整型和无符号短整型两种不同的数据类型存储。在图11中，我们比较了10个稀疏矩阵采用两种FCSR存储格式存储时SpMV运行1000次的时间，两种方式使用的向量x的访存方式一致，且都做了双缓冲优化。结果显示，*Aindex*采用无符号短整数类型存储相比*Aindex*采用整数类型存储，SpMV的最高加速比为1.10，平均加速比为1.04。后续的测试将采用*Aindex*为无符号短整数类型的FCSR存储格式进行测试。

我们测试了10个稀疏矩阵从CSR存储格式转换为FCSR存储格式所需的时间，并将其与优化后的SpMV运行时间进行了对比，具体结果如图12所示。从CSR存储格式到FCSR存储格式的转换时间与一次SpMV运行时间的比值在10到100之间，

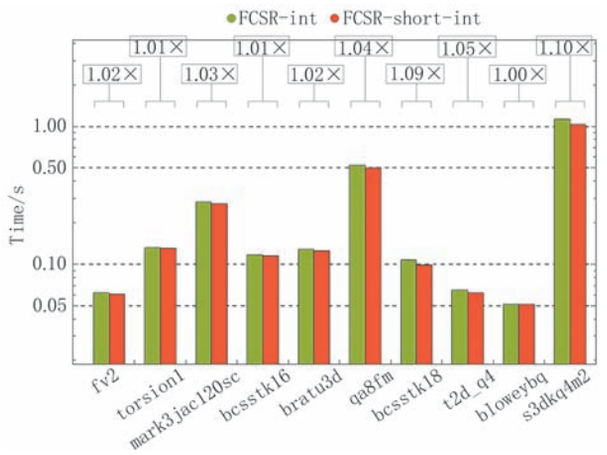

图11 Aindex不同存储方式的SpMV运行时间对比

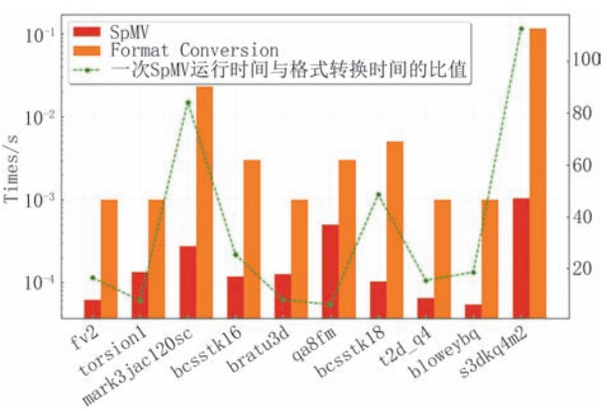

图12 格式转换时间与SpMV运行时间对比

即测试矩阵中一次格式转换所需的时间最多相当于100次SpMV的运行时间。格式转换会在实际应用计算之前一次完成，尽管初始的格式转换会带来一定的开销，但实际应用中SpMV计算通常需要进行数千次甚至上万次的迭代，格式转换的开销在整个计算过程中所占的比例相对较小。

为了验证本文提出的新型存储格式的有效性，我们对比测试了14个稀疏矩阵基于传统CSR存储格式的SpMV算法与采用无符号短整型的FCSR存储格式的SpMV算法运行1000次的时间，两种算法采用的向量的访存方式一致，且都做了双缓冲优化。测试结果如图13所示，基于FCSR存储格式的SpMV算法相较于基于CSR存储格式的SpMV算法，最高加速比达到了1.64，平均加速比为1.19。

5.5 整体效果评估

我们在SuiteSparse矩阵集中选取了508个规模较大的稀疏矩阵，对比测试了这些矩阵基于CSR存储格式的SpMV算法与基于*Aindex*采用无符号短整型的FCSR存储格式的SpMV算法运行1000次的时间，两种方式采用的向量x访存方式一致，且都

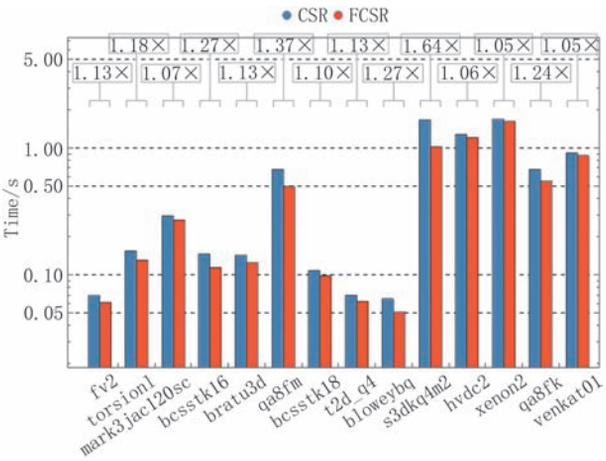

图13　基于CSR与FCSR存储格式的SpMV运行时间对比

做了双缓冲优化。图14展示了两种算法的加速比随矩阵可压缩率的变化情况，可以看出，对于大多数矩阵，当矩阵可压缩率大于0.5时，基于FCSR存储格式的SpMV算法相较于基于CSR存储格式的SpMV算法具有较明显的性能提升，最高加速比可达1.55。这一结果验证了第3.3节中的观点，即当矩阵可压缩率满足$\alpha > 0.5 + 32/nz$时，选用FCSR存储格式下的SpMV具有访存优势。

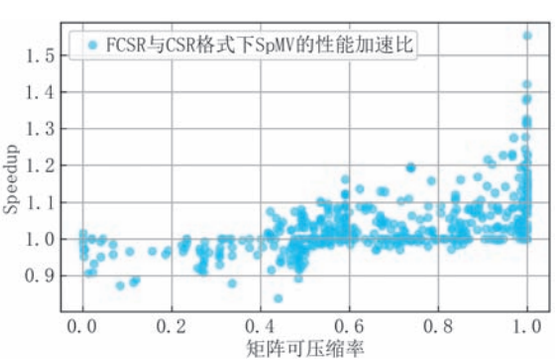

图14　基于FCSR与CSR存储格式的SpMV算法加速比随矩阵可压缩率的变化图

我们还对比了508个稀疏矩阵基于*Aindex*采用无符号短整型的FCSR存储格式的SpMV算法与主核版SpMV算法运行1000次的时间。图15为这两种算法的加速比情况，结果表明，随着矩阵非零元数量的增加，采用FCSR存储格式的异构众核SpMV算法相比于主核版本SpMV算法的加速效果越发显著，最高加速比达到了43.11，平均加速比为7.56。然而，少数非零元数量较多的矩阵未表现出明显的加速效果，这主要是由于这些矩阵中非零元素分布不均匀，导致并行计算优势无法充分发挥。

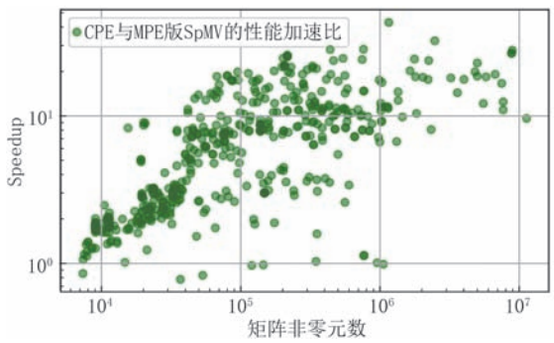

图15　基于FCSR存储格式的异构众核SpMV与主核版SpMV的性能对比

通过公式(6)可以计算出*Aindex*元素采用无符号短整型存储时SpMV的总访存量*SFCSR*。基于矩阵的总访存量和一次SpMV的运行时间t，可以推导出这些矩阵进行SpMV计算时的访存带宽*bandwith*：

$$bandwith = \frac{SFCSR/10^9}{t} \tag{10}$$

单个核组主存的实测访存带宽约为46 GB/s，则带宽利用率ω为

$$\omega = bandwith/46 \tag{11}$$

508个测试矩阵的带宽利用率情况如图16所示，可以看出，随着矩阵非零元素数量的增加，带宽利用率也随之增加。最高带宽利用率可达到91.13%，平均带宽利用率为26.27%。图中少数非零元素数量较多的矩阵在带宽利用率方面表现不佳，这主要是由于非零元素分布不均造成的。

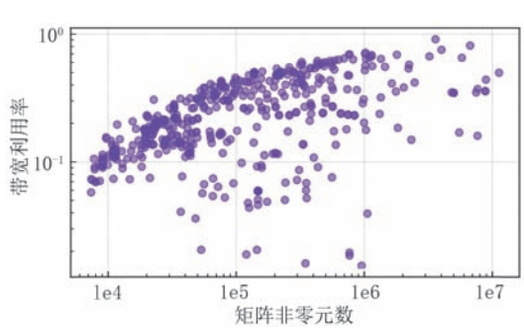

图16　基于FCSR存储格式的SpMV带宽利用率

在本文提出的计算SpMV总访存量的方法中，假设向量x可以完全重用，即将向量x的总访存次数为矩阵的维数n。然而，在实际计算中，一个向量x可能需要被多次访问，实际访问次数远远超过n，因此本文计算的带宽利用率为实际带宽利用率的最小值。

6 总 结

本文对CSR存储格式中的非零元素数组进行了进一步的压缩，提出了一种新型的稀疏矩阵存储格式FCSR，并设计了从CSR到FCSR格式转换的异构并行算法。同时，针对SW26010-Pro众核处理器，本文设计了基于FCSR存储格式的SpMV异构并行算法，对SpMV进行了细粒度的任务划分和并行优化设计，探究了五种向量x的间接访存方式，并通过双缓冲技术对算法进行了优化。实验结果表明，通过“Cache Read”方式获取向量x时SpMV性能表现最佳，双缓冲技术可有效提升SpMV性能。本文对比了基于FCSR存储格式与基于CSR存储格式的SpMV算法的性能，在两者都充分优化的前提下，平均加速比为1.19。基于FCSR存储格式的异构众核SpMV算法相较于主核版本最高加速比达到43.11，平均加速比为7.56，测试矩阵最高带宽利用率达到了91.13%，平均带宽利用率为26.27%。

本文提出的FCSR存储格式及其格式转换算法不局限于特定平台，可以应用于多种主流的多/众核处理器。同时，本文提出的基于FCSR存储格式的SpMV异构并行算法可在SW26010等类似异构处理器上实现移植应用，其中，多种向量x的间接访存方式在不同平台上的适用性有所不同，应用时需根据实际情况选择。在自然语言处理、深度学习以及图像处理等诸多领域，SpMV计算都有着广泛的应用，针对这些应用场景下具有较高可压缩率的矩阵，可考虑采用FCSR格式进行存储和SpMV计算。另外，随着越来越多的人工智能问题向SW26010-Pro等异构平台迁移，本文提出的SpMV优化方案对于解决这类问题也具有重要参考意义。

针对基于FCSR存储格式的SpMV算法，未来的研究需要考虑实现更优质的负载均衡方法，例如动态任务划分和更细粒度任务调度等。此外，未来的研究还将探索混合精度在基于FCSR格式的SpMV算法中的应用，从不同的角度进一步提升算法整体性能。

致 谢 感谢所有评审人员的建议和帮助!

参 考 文 献

[1] Narasimharao N , Mallaiah A . VLSI Design of area efficient high performance SPMV accelerator using VBW-CBQCSR scheme. International Journal of Advanced Research in Computer and Communication Engineering, 2016, 5(8), 213-217

[2] Fu H, He C, Chen B, et al. 9-Pflops nonlinear earthquake simulation on Sunway TaihuLight: Enabling depiction of 18-Hz and 8-meter scenarios//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. Denver Colorado, USA , 2017: 1-12

[3] Isotton G, Frigo M, Spiezia N, et al. Chronos: A general purpose classical AMG solver for high performance computing. SIAM Journal on Scientific Computing, 2021, 43(5): C335-C357

[4] Courbariaux M, Hubara I, Soudry D, et al. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv: 1602.02830, 2016

[5] Qiu S, You L, Wang Z. Optimizing sparse matrix multiplications for graph neural networks//International Workshop on Languages and Compilers for Parallel Computing. Newark,USA, 2021: 101-117

[6] Foss A H, Lehoucq R B, Stuart W Z, et al. A deterministic hitting-time moment approach to seed-set expansion over a graph. arXiv preprint arXiv:2011.09544, 2020

[7] Diaz J, Munoz-Caro C, Nino A. A survey of parallel programming models and tools in the multi and many-core era. IEEE Transactions on Parallel and Distributed Systems, 2012, 23(8): 1369-1386

[8] Chrysos G. Intel® xeon phi ™ coprocessor-the architecture. Intel Whitepaper, 2014, 176(2014): 43-50

[9] Wu R, Zhu X, Chen J, et al. SWattention: Designing fast and memory-efficient attention for a new Sunway Supercomputer. The Journal of Supercomputing, 2024, 80(10): 13657-13680

[10] Chou S, Kjolstad F, Amarasinghe S. Format abstraction for sparse tensor algebra compilers. Proceedings of the ACM on Programming Languages, 2018, 2(OOPSLA): 1-30

[11] Gao J, Ji W, Chang F, et al. A systematic survey of general sparse matrix-matrix multiplication. ACM Computing Surveys, 2023, 55(12): 1-36

[12] Williams S, Oliker L, Vuduc R, et al. Optimization of sparse matrix-vector multiplication on emerging multicore platforms// Proceedings of the 2007 ACM/IEEE Conference on Supercomputing. Reno Nevada, USA, 2007: 1-12

[13] Bian H, Huang J, Liu L, et al. Albus: A method for efficiently processing spmv using simd and load balancing. Future Generation Computer Systems, 2021, 116: 371-392

[14] Kourtis K, Goumas G, Koziris N. Optimizing sparse matrix-vector multiplication using index and value compression// Proceedings of the 5th Conference on Computing Frontiers. Ischia, Italy , 2008: 87-96

[15] Kourtis K, Karakasis V, Goumas G, et al. CSX: An extended compression format for spmv on shared memory systems. ACM SIGPLAN Notices, 2011, 46(8): 247-256

[16] Tang W T, Zhao R, Lu M, et al. Optimizing and auto-tuning

scale-free sparse matrix-vector multiplication on Intel Xeon Phi//Proceedings of the 2015 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). San Francisco, USA, 2015: 136-145

[17] Xie B, Zhan J, Liu X, et al. Cvr: Efficient vectorization of spmv on x86 processors//Proceedings of the 2018 International Symposium on Code Generation and Optimization. Vienna, Austria ,2018: 149-162

[18] Bian H, Huang J, Dong R, et al. A simple and efficient storage format for SIMD-accelerated SpMV. Cluster Computing, 2021, 24: 3431-3448

[19] Fukaya T, Ishida K, Miura A, et al. Accelerating the SpMV kernel on standard CPUs by exploiting the partially diagonal structures. arXiv preprint arXiv:2105.04937, 2021

[20] Nathan B and Michael G. Implementing sparse matrix-vector multiplication on throughput-oriented processors//Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. Portland,USA,2009,pp. 1-11.

[21] Zhao Z, Zhang G, Wu Y, et al. Block-wise dynamic mixed-precision for sparse matrix-vector multiplication on GPUs. Journal of Supercomputing, 2024, 80:13681-13713

[22] Yang W, Li K, Li K. A parallel computing method using blocked format with optimal partitioning for SpMV on GPU. Journal of computer and system sciences, 2018, 92: 152-170

[23] Choi J W, Singh A, Vuduc R W. Model-driven autotuning of sparse matrix-vector multiply on GPUs. ACM sigplan notices, 2010, 45(5): 115-126

[24] Liu Y, Schmidt B. LightSpMV: Faster CSR-based sparse matrix-vector multiplication on CUDA-enabled GPUs// Proceedings of the 2015 IEEE 26th International Conference on Application-specific Systems, Architectures and Processors (ASAP). Toronto, Canada, 2015: 82-89

[25] Daga M, Greathouse J L. Structural agnostic SpMV: Adapting CSR-adaptive for irregular matrices// Proceedings of the 2015 IEEE 22nd International Conference on High Performance Computing (HiPC). Bengaluru, India, 2015: 64-74

[26] Cui H, Wang N, Han Q, et al. A two-stage parallel method on GPU based on hybrid-compression-format for diagonal matrix. Concurrency and Computation: Practice and Experience, 2024, 36(1): e7887

[27] Zhang Y, Li S, Yan S, et al. A cross-platform SpMV framework on many-core architectures. ACM Transactions on Architecture and Code Optimization (TACO), 2016, 13(4): 1-25

[28] Liu W, Vinter B. CSR5: An efficient storage format for cross-platform sparse matrix-vector multiplication // Proceedings of the 29th ACM on International Conference on Supercomputing. California, USA, 2015: 339-350

[29] Coronado-Barrientos E, Indalecio G, García-Loureiro A. AXC: A new format to perform the SpMV oriented to Intel Xeon Phi architecture in OpenCL. Concurrency and Computation: Practice and Experience, 2019, 31(1): e4864

[30] Chen C. Explicit caching HYB: A new high-performance SpMV framework on GPGPU. arXiv preprint arXiv:2204.06666, 2022

[31] Wang Yu-Hua, Zhang Yu-Qi, He Jun-Fei, et al. TEB: Efficient SpMV storage format for matrix decomposition and reconstruction on GPU. Journal of Frontiers of Computer Science and Technology, 2024, 18(4): 1094-1108 (in Chinese)
(王宇华,张宇琪,何俊飞等.TEB:GPU上矩阵分解重构的高效SpMV存储格式.计算机科学与探索, 2024, 18(4):1094-1108)

[32] King J, Gilray T, Kirby R M, et al. Dynamic sparse-matrix allocation on GPUs//Proceedings of the 31st International Conference of High Performance Computing, Frankfurt, Germany, 2016: 61-80

[33] Du Z, Li J, Wang Y, et al. Alphasparse: Generating high performance spmv codes directly from sparse matrices//SC22: International Conference for High Performance Computing, Networking, Storage and Analysis. Dallas, USA, 2022: 1-15

[34] Liu Fang-Fang, Yang Chao, Yuan Xin-Hui, et al. General SpMV implementation in many-core domestic sunway 26010 processor. Journal of Software, 2018, 29(12): 3921-3932 (in Chinese)
(刘芳芳, 杨超, 袁欣辉等. 面向国产申威 26010 众核处理器的 SpMV 实现与优化. 软件学报, 2018, 29(12): 3921-3932)

[35] Li Yi-Yuan, Xue Wei, Chen De-Xun, et al. Performance optimization of sparse matrix-vector multiplication on Sunway architecture. Chinese Journal of Computers, 2020, 43(6), 1010-1024 (in Chinese)
(李亿渊, 薛巍, 陈德训等. 稀疏矩阵向量乘法在申威众核架构上的性能优化. 计算机学报, 2020, 43(6), 1010-1024)

[36] Ma M, Huang X, Xu J, et al. Implementation and optimization of SpMV algorithm based on SW26010P many-core processor and stored in BCSR format. Scientific Reports, 2024, 14(1): 16574

[37] Pan J, Xiao L, Tian M, et al. hsSpMV: A heterogeneous and SPM-aggregated SpMV for SW26010-Pro many-core processor// Proceedings of the 2023 IEEE/ACM 23rd International Symposium on Cluster, Cloud and Internet Computing (CCGrid). Bangalore, India, 2023: 62-70

WANG Cui M. S., engineer. Her research interests include high-performance computing, heterogeneous parallelism, and sparse matrix-related algorithm research.

LIU Fang-Fang, Ph. D., senior advanced engineer. Her research interests include high-performance computing, heterogeneous parallelism, and supercomputer evaluation software.

MA Wen-Jing, Ph. D., assistant professor. Her research interests include high-performance computing, code generation and optimization.

ZHAO Yu-Wen, Ph. D., senior engineer. Her research interests include high-performance computing, heterogeneous parallelism, and FFT-related algorithm research.

HU Li-Juan, M. S., assistant engineer. Her research interests include high-performance computing, heterogeneous parallelism, and BLAS library related algorithm research.

Background

Sparse matrix-vector multiplication (SpMV) plays a vital role in science and engineering, and is widely used in different fields such as meteorological simulation, signal processing, fluid mechanics, and machine learning. With the continuous increase in the scale of problems and the continuous evolution of hardware structures, the optimization of SpMV faces challenges such as bandwidth limitation and load imbalance.

Efficient sparse matrix storage format is an important factor affecting the performance of SpMV. However, the existing storage format mainly compresses zero elements to reduce memory access, and does not fully utilize the numerical regularity of non-zero elements. Therefore, there is still room for further compression and optimization. On the other hand, SW26010-Pro is a new generation of Shenwei heterogeneous multi-core processor independently developed by my country. Compared with the SW26010 processor, its hardware characteristics have been greatly improved. The existing optimization scheme on SW26010 is not enough to give full play to the hardware characteristics of SW26010-Pro.

First, based on the CSR storage format, this paper further compresses the non-zero element part and proposes a new sparse matrix storage format FCSR. At the same time, we designed a heterogeneous parallel algorithm for converting from CSR to FCSR format to reduce the overhead in the conversion process. The format conversion algorithm has good portability and can be applied on various multi-core/many-core platforms.

At the same time, this paper deeply analyzes the characteristics of the domestic SW26010-Pro platform, proposes a SpMV parallel algorithm based on the FCSR storage format, performs fine-grained task division and parallel optimization design on SpMV, explores various indirect memory access methods of vector x, and further optimizes the algorithm using double buffering technology. The heterogeneous many-core SpMV algorithm based on FCSR storage format proposed in this paper has significantly improved performance compared with the controller corevision; it also has obvious performance compared with the heterogeneous many-core SpMV algorithm based on CSR storage format with the same optimization technology. The SpMV heterogeneous parallel algorithm based on FCSR storage format proposed in this paper can be implemented on similar heterogeneous hardware platforms such as SW26010.

This work was supported by the National Key R&D Program (2023YFB3001703) and the self-developed project of the Institute of Software, Chinese Academy of Sciences (ISCAS-JCMS-202304). It was used to solve the sparse matrix-vector multiplication problem in the project.