计 算 机 研 究 与 发 展
Journal of Computer Research and Development
https://crad.ict.ac.cn
63(11): 2807−2821, 2026

面向数据流架构的稀疏矩阵高效处理内核

蒋文斌　刘宝阳　董雨康　沈欣海
（大数据技术与系统国家地方联合工程研究中心（华中科技大学）　武汉　430074）
（服务计算技术与系统教育部重点实验室（华中科技大学）　武汉　430074）
（集群与网格计算湖北省重点实验室（华中科技大学）　武汉　430074）
（华中科技大学计算机科学与技术学院　武汉　430074）
（wenbinjiang@hust.edu.cn）

Efficient Sparse Matrix Multiplication Kernel for Dataflow Architecture

Jiang Wenbin, Liu Baoyang, Dong Yukang, and Shen Xinhai
（*National Engineering Research Center for Big Data Technology and System* (*Huazhong University of Science and Technology*), *Wuhan* 430074）
（*Services Computing Technology and System Lab* (*Huazhong University of Science and Technology*), *Ministry of Education*, *Wuhan* 430074）
（*Cluster and Grid Computing Lab* (*Huazhong University of Science and Technology*), *Wuhan* 430074）
（*School of Computer Science and Technology*, *Huazhong University of Science and Technology*, *Wuhan* 430074）

Abstract　In recent years, sparse-dense matrix multiplication (SpMM) has become a core operator in various fields such as scientific computing, graph processing, and deep learning, while also being a bottleneck limiting system performance. However, existing GPU implementations of SpMM generally suffer from low energy efficiency and resource utilization. Specialized accelerators demonstrate excellent energy efficiency in regular computations, while lack efficient support for irregular sparse matrix operations. To this end, we propose an efficient sparse matrix processing kernel tailored for dataflow architecture, DFU-SpMM, targeting the high-throughput dataflow unit (DFU). Firstly, a hierarchical sparse matrix processing method is proposed to ensure that the sparse blocks have a regular calculation pattern to reduce the number of dataflow graph retransmissions on the SpMM kernel on DFU. Meanwhile, for sparse blocks with different characteristics, different strategies are used to process them. Secondly, an adaptive assembly code generation method for dataflow graph optimization is proposed for optimizing dataflow graph. By leveraging the inspector-executor framework to overcome the current limitations of DFU in generating SpMM kernels, the dataflow graph is further optimized during code generation with respect to task allocation and register reuse. Experiments on representative datasets demonstrate that, compared with state-of-the-art implementations DTC-SpMM and HP-SpMM on an RTX 4090 GPU, the proposed method achieves an average 1.09 times and 1.23 times higher energy efficiency, along with 1.71 times and 1.10 times better computational resource utilization, respectively. Compared with the baseline SpMM kernel on the DFU accelerator without data reorganization, the optimized kernel with data reorganization achieves 1.41 times speedup and 1.51 times energy efficiency improvement.

Key words　deep learning; high-performance computing; sparse computation acceleration; dataflow graph; data reorganization

收稿日期: 2025-09-21; **修回日期**: 2026-06-09

基金项目: 国家重点研发计划项目(2022YFB4501400); 国家自然科学基金项目(62372199、U25B2023); 教育部基础学科和交叉学科突破计划项目(JYB2025XDXM118); 湖北省自然科学基金卓越研究群体项目(2026AFA002)

This work was supported by the National Key Research and Development Program of China (2022YFB4501400), the National Natural Science Foundation of China (62372199、U25B2023), the Fundamental and Interdisciplinary Disciplines Breakthrough Plan of the Ministry of Education of China (JYB2025XDXM118), and the Hubei Provincial Natural Science Foundation of China (2026AFA002).

摘 要 近年来,稀疏矩阵-密集矩阵乘(SpMM)已成为科学计算、图处理和深度学习等多个领域的核心算子和制约系统性能的瓶颈。然而,现有通用GPU上的SpMM实现普遍存在能效比、资源利用率低的问题,而一些专用的加速器虽然在规则计算上展现出了优异的能效比,但缺乏对不规则的稀疏矩阵运算的高效支持。针对此问题,基于高通量数据流众核处理器(DFU),提出了一种面向数据流架构的稀疏矩阵高效处理内核DFU-SpMM。首先提出数据流指令复用的分层稀疏矩阵处理方法,对稀疏数据进行重组织以减少SpMM内核的数据流图重传次数,同时,对于不同稀疏块使用不同的策略对其进行处理。然后提出面向流图优化的自适应汇编代码生成方法,引入观察者-执行者的思想,以突破现有DFU在SpMM内核生成方面的能力限制,并从任务分配和寄存器重用2个角度对代码生成过程中的数据流图构造进行优化。代表性数据集上的实验结果表明:相较于RTX 4090 GPU环境下的现有前沿工作DTC-SpMM和HP-SpMM,DFU-SpMM分别提高了平均1.09倍与1.23倍的能效以及1.71倍与1.10倍的计算资源利用率;相较于DFU加速器上数据重组织前的SpMM内核,优化后的内核实现了1.41倍的性能提升以及1.51倍的能效比提升。

关键词 深度学习;高性能计算;稀疏计算加速;数据流图;数据重组织

中图法分类号 TP332

DOI: 10.7544/issn1000-1239.202550678 **CSTR:** 32373.14.issn1000-1239.202550678

随着深度学习的快速发展,以GPT-3[1],BERT[2]为代表的大规模预训练模型(大模型)在人工智能领域取得重要突破。受益于其海量的模型参数(千亿级甚至万亿级)以及注意力机制[3],大模型展现出强大的语言理解、生成和推理能力,被广泛应用于文本翻译[4]、图片生成[5]等领域,受到各界的广泛关注。

然而大模型强大的表达能力也伴随着对系统资源和能耗的庞大需求,需要大量的高性能处理器以及高昂的电力成本[6],通常只有大企业及研究机构能够参与其核心研究。为了缓解大模型带来的资源压力,现有研究设计各种不同的模型剪枝[7]算法将大模型的参数稀疏化,进而产生了稀疏注意力机制[8],这使得稀疏矩阵-密集矩阵乘(sparse matrix-dense matrix multiplication, SpMM)成为大模型的核心操作及性能瓶颈。

此外,图神经网络(graph neural network, GNN)对图结构中节点以及节点间的连接关系进行建模,具有强大的非欧几里得数据处理能力,在图数据上展现出很强的表达能力,目前已在推荐系统[9]、蛋白质结构分析[10]、知识图谱[11]等领域展现出显著优势。但现实场景中的图数据通常具有庞大的数据规模(如社交网络中用户的连接关系),图神经网络也对算力和能耗有着极高的需求。图神经网络通常可以分为聚合和更新2个阶段,由于现实中图数据具有天然的稀疏性,前一阶段通常抽象为SpMM内核,后一阶段根据模型中权重矩阵的稀疏度,通常抽象为密集矩阵-密集矩阵乘(general purpose matrix multiplication, GEMM)内核或SpMM内核。现有研究表明,SpMM内核作为图神经网络的核心计算操作,其执行效率已成为制约模型训练速度和推理延迟的关键瓶颈[12]。

因此,优化SpMM内核的计算效率对提升大模型和图神经网络的推理能力和训练速度至关重要。现有SpMM内核优化的相关研究主要集中在GPU上,通过利用GPU提供的多级缓存结构和高效内存访问模式加快内核执行过程的访存效率,以达到提高SpMM内核性能的目的。英伟达(NVIDIA)公司推出的GPU是一种用来处理图形和并行计算任务的通用加速器(以下提到的GPU都特指NVIDIA公司的GPU)。GPU提供大量的并行计算核心以及高效的内存带宽,同时其单指令多线程(SIMT)的设计让多个线程能够同时执行同一指令而处理不同数据,减少了控制流架构中指令译码等开销,这使其在处理密集矩阵运算等能够高度并行且访存地址连续的任务时具有很高的吞吐量。除此之外,GPU提供的以C语言为基础的编程模型(CUDA)及其良好的软件生态让开发者能够方便高效地编写CUDA程序对内核进行优化,并调用GPU进行计算,充分利用GPU的计算和存储资源,这些因素让GPU成为深度学习研究领域的主流加速器。

尽管GPU提供了强大的峰值算力,但大量计算核心的高度并行也带来了很高的运行时功耗,且频繁的线程调度和低效的显存访问使GPU在实际任务中的表现高度依赖于对计算核心和多级缓存的利用程度。数据流架构的加速器与基于控制流架构的

GPU 不同，其通过片上数据流动和数据驱动的方式减少执行过程中对低效内存的访问以及指令译码、分支预测等指令开销，且硬件级流水线使其在执行任务时通常能够更有效地减少硬件资源的空闲时间，进而获得优于控制流架构处理器的能效比和计算资源利用率，代表性工作有谷歌（Google）公司的张量处理单元（tensor processing unit, TPU）[13] 和 Wave Computing 公司的数据流处理单元（dataflow processing unit, DPU）等，这为提高 SpMM 内核性能提供新的优化方向。

然而，稀疏矩阵的不规则数据分布带来的随机内存访问和计算资源负载不均衡等问题使普适性的加速器在稀疏计算任务中很难充分发挥其硬件性能。为了解决通用加速器难以适应稀疏计算特性的问题，各种针对稀疏计算任务的专用加速器[14-16] 应运而生，它们通常通过设计专用的硬件来加速计算过程，如特斯拉（Tesla）公司推出的超级计算机 Dojo 利用加速器上专用的硬件任务队列来自动平衡长短行计算，进而获得远高于 GPU 的计算资源负载均衡度；英特尔（Intel）公司设计的人工智能加速器 Habana 通过引入专用的稀疏内存引擎来提高随机内存访问效率等。尽管这些加速器在特定领域相较通用加速器具有显著优势，但专用的硬件设计也限制了其泛用性，需要为具有不同特性的计算任务设计特殊的硬件，这使其通常依靠频繁更新硬件架构来适应不断出现的先进算法，具有很高的硬件迭代成本。同时专用加速器的软件生态通常不够成熟，这带来了极大的软件开发和部署成本。

中国科学院推行的高通量数据流众核处理器（DFU）是一种面向多应用的通用数据流架构加速器[17-18]，不同于专用加速器的是，其没有通过设计专用的硬件来解决特定领域任务面临的挑战，而是通过灵活配置数据流图来改变片上执行单元间的数据流动关系，以适应不同任务的计算特性，同时，DFU 中数据流图复用的机制有效减少了数据流图配置开销。

尽管 DFU 已在数字信号处理、卷积神经网络以及科学计算等领域展现出优于 GPU 及其他数据流加速器的显著优势，但同样面临着通用加速器难以高效处理稀疏计算的问题。受制于其硬件架构和编程模型的设计，DFU 数据流加速器上的编译器（DFUC）目前不支持稀疏计算内核的编译，且 SpMM 内核的不规则内存访问使其处理数据时所需的汇编指令和节点间的数据流动关系改变，进而导致部署在加速器上的数据流图频繁变化，难以发挥 DFU 加速器上数据流图重用和执行单元间数据流动等核心特性，如何设计一种 DFU 上数据流友好的稀疏矩阵高效处理内核是一个亟待解决的问题。

针对上述问题，本文重点研究高效的数据重组织算法，将不规则的稀疏矩阵在 DFU 的片上存储空间中规则化，以提高 SpMM 内核执行过程中数据流图的重用率。同时，根据稀疏矩阵的数据分布直接生成汇编代码，并结合 DFU 加速器特性，从多角度探索代码生成策略，确保内核能够充分利用 DFU 加速器的硬件资源，以达到数据流友好的目的。

1 相关工作

1.1 GPU 上 SpMM 优化

现有研究对 SpMM 内核在软件层面上的优化集中在 GPU 上，主要包括稀疏矩阵预处理和内核运算过程优化 2 方面。稀疏矩阵预处理通过改变稀疏矩阵的数据分布来更好地利用 GPU 的多级缓存或特殊的矩阵运算加速单元。而内核运算过程优化则是在预处理的基础上利用 GPU 上用户可控的高效存储资源和内存访问模式提高 SpMM 内核运算时的访存效率，同时通过合理的工作负载分配来减少计算单元的空闲时间。

1）稀疏矩阵预处理

稀疏矩阵的不规则数据分布使其具有很差的数据局部性，难以有效利用 GPU 的多级缓存结构，通常需要预先对稀疏矩阵进行重排序来提高 GPU 的缓存命中率。Jiang 等人[19] 提出一种基于相似度的局部敏感哈希（locality sensitive hashing, LSH）重排序算法，该算法根据稀疏矩阵行上非零元数目计算稀疏行间的杰卡德相似度（Jaccard similarity），并根据相似度将稀疏行划分到不同的桶中，每个桶中维护 1 组相似的稀疏行，然后使用并查集算法对每个桶中的稀疏行进行聚类，使相似的稀疏行在稀疏矩阵中尽可能相邻，有效提高稀疏矩阵的数据局部性。尽管其加速了相似度的计算，但其难以在 GPU 上并行，仍具有很高的时间开销。为了解决这一问题，HP-SpMM[20] 采用另一种基于相似度的重排序算法对稀疏矩阵进行数据重组织，其重排序过程能够在 GPU 上高效执行，有效降低基于相似度的重排序算法带来的高昂时间成本。

尽管上述基于相似度的重排序算法通过改善稀疏矩阵的数据局部性来提高 GPU 的缓存命中率，但并没有考虑 SpMM 内核计算过程中计算资源负载不

均衡的问题。Accel-GCN[21]采用一种混合负载均衡优化策略,基于稀疏矩阵行上非零元数目对使用压缩稀疏行(compressed sparse row, CSR)格式存储的稀疏矩阵进行计数排序,使重排序算法的整体时间复杂度达到 $O(n)$ 级别,显著降低了重排序算法的时间开销,并在此基础上结合按行分配及按非零元数目分配 2 种任务分配方式,在 SpMM 内核的线程块和线程束 2 个层面上进行负载均衡优化,减少 GPU 上计算资源的空闲时间。

此外,TC-GNN[22]对稀疏矩阵的行进行预分块处理并将稀疏块中的非零元列号去重,以压缩稀疏块来增加矩阵运算单元处理的稀疏块的非零元密度,减少 GPU 上专用矩阵运算核心(tensor core)的启动次数进而改善 SpMM 内核性能。

2)内核运算过程优化

稀疏矩阵中存在着大量零元,直接使用邻接矩阵存储稀疏数据会浪费大量存储资源,同时带来大量无效计算。DTC-SpMM[23]针对 GPU 上的 tensor core 具有单次处理的稀疏矩阵块的尺寸为 128(16×8 的数据规模)的硬件特性,使用 8 b 整数来表示稀疏数据在矩阵块中的位置信息,进一步压缩稀疏矩阵。

SparseTIR[24]利用稀疏矩阵的不同部分更适合不同存储格式的特性,设计可组合多种稀疏存储格式的编译器,构造编译前提供的可组合稀疏存储格式的搜索空间,自动为矩阵的不同区域分别选择合适的稀疏存储格式来压缩稀疏矩阵,并生成对应的 SpMM 内核,充分利用不同稀疏存储格式的特性。

除稀疏存储格式优化外,如何合理利用 GPU 上高效的存储资源也是提高 SpMM 内核性能的关键。GE-SpMM 通过将具有数据依赖的循环按照 32 的长度部分展开(loop unrolling),通过合并内存访问将稀疏数据预取到 GPU 的共享内存中,来减少后续计算中对低效的全局内存的访问。SparseRT[25]利用观察者-执行者框架获取稀疏数据信息并进行编译时代码嵌入,利用 GPU 的常量缓存提高访问稀疏数据的效率。

1.2 加速器上 SpMM 优化

数据流架构加速器数据驱动的特性消除了指令译码、分支预测等指令开销,同时其独特的片上数据流动能够有效减少对低效内存的访问,但其难以处理稀疏矩阵,SIGMA[26]通过多播机制将需要的密集数据一次性传输到所有可使用的处理单元上,以解决脉动阵列固定的任务分配方式在处理稀疏数据时大量计算单元空闲的问题。ALRESCHA[27]设计一种由连接固定的并行处理单元和连接可重配置的归约处理单元构成的数据流加速器,并为无数据依赖的部分和有数据依赖的部分分别使用不同的处理单元进行处理以利用稀疏矩阵不同部分的特性。

相较上述数据流加速器,DFU 能够对硬件进行更灵活的控制,根据不同内核的特性构造不同的数据流图来控制计算资源间的数据流动以提高数据重用,且其数据流图重用的特性能够有效减少数据流图配置的开销。其已在部分图计算任务如广度优先搜索(breadth-first search, BFS)等上有效提高硬件能效,但目前无法适配 SpMM 内核。

除了数据流架构的加速器外,现有研究还针对 SpMM 内核的特性设计专用加速器,这些加速器往往通过特殊的硬件来加速内核计算过程中的一部分,并通过硬件流水线提高计算资源的利用率。Gao 等人[28]中提出一种软硬件协同优化的 SpMM 内核,在现场可编程门阵列(field-programmable gate array, FPGA)上对稀疏数据进行非零元数分配以实现负载均衡,并通过板卡上的缓存为 SpMM 计算过程提供硬件流水线以减少同步开销。GRIP[29]提供多级预取和归约模块实现硬件流水线以加速 SpMM 内核。

2 分析与动机

DFU 的硬件架构如图 1 所示,其主体由 16 个可并行执行的处理单元(processing unit, PE)及 1 个由全部 PE 共享的用户管理型片上暂存存储器(scratchpad memory, SPM)构成。主机端采用基于 RISC-V 核的处理器,DFU 作为协处理器与之协同工作。主机通过直接内存访问(direct memory access, DMA)机制实现主存与片上 SPM 间的数据传输,该过程由内存控

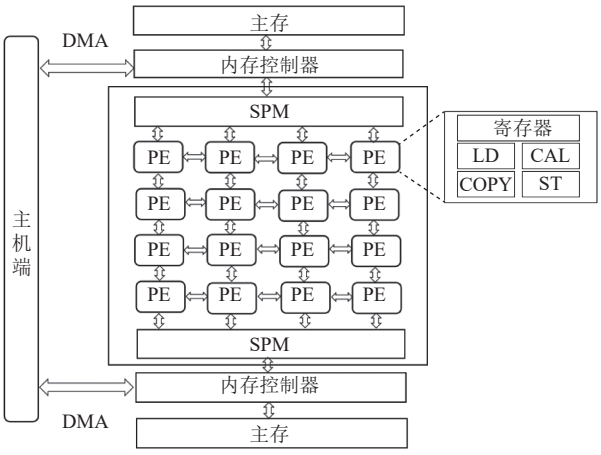


Fig. 1 Hardware architecture of DFU accelerator

图 1 DFU 加速器硬件架构

制器统一调度与管理。

DFU 加速器内部的 16 个 PE 由片上网络互联，并通过网络进行 PE 之间的数据流动以实现数据复用。每个 PE 内部包括 1 组寄存器和 4 个功能部件，其中，加载功能部件(LD)负责从 SPM 缓存加载数据到寄存器，复制功能部件(COPY)负责控制 PE 之间的数据流动，计算功能部件(CAL)负责执行计算操作，存储功能部件(ST)负责将寄存器中的数据存储到 SPM 缓存。

DFU 根据部署在其上的数据流图运行，数据流图的节点为不同的汇编指令，汇编指令主要分为 4 类，分别控制并行的 4 个功能部件。在 2 次 PE 阵列启动间，若数据流图未发生变化，则不必重新传输。在规则计算(如密集矩阵-密集矩阵乘)中，每次内核所需访问的 SPM 缓存地址相同，数据流图不会发生变化，故其通常仅在首次阵列启动和末次阵列启动时传输数据流图以减少流图配置开销。

SpMM 内核在处理不同稀疏数据时会产生不同于密集矩阵-密集矩阵乘的数据流图，如图 2 所示。稀疏矩阵的不规则分布使 SpMM 内核需要根据列号访问不同的密集矩阵数据，这导致了访存指令对应的 SPM 缓存地址发生变化，同时其数据重用依赖于矩阵的稀疏结构，这导致流图节点间的数据流动具有不确定性，难以重用固定的数据流图，进而产生了大量的数据流图配置开销。为此，频繁的流图重传是 DFU 加速器处理稀疏矩阵不友好的一个重要的原因。

此外，DFU 加速器上的编译器(DFUC)不支持 SpMM 内核的编译，为了使 DFU 加速器能够进行稀疏矩阵处理需要手动编写汇编指令，这带来了极高的开发成本，且其无法利用编译器提供的流图优化，难以确保内核的高效性。同时加速器未提供内核执行时所必须的间接寻址相关指令且内核执行时仅能访问 SPM 缓存，无法直接访问主机端上的 DRAM 主存，而 SPM 缓存仅有 6 MB，为此，如何绕过 DFUC 编译器生成汇编指令并在此基础上尽可能优化数据流图与主存访问，是稀疏矩阵高效处理内核的关键。

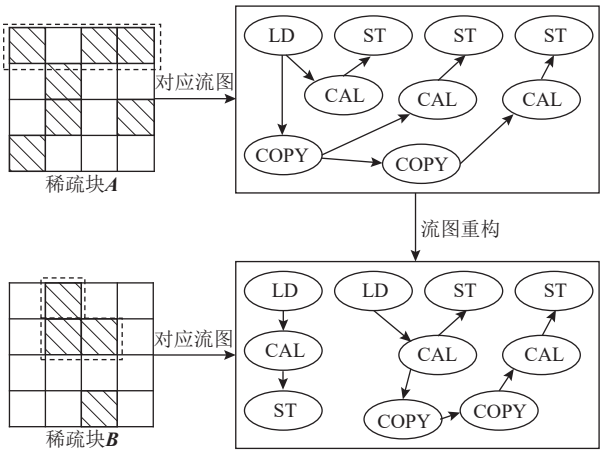


Fig. 2 Dataflow graph reconstruction of sparse matrix processing

图 2 稀疏矩阵处理过程中的数据流图重构

3 面向数据流架构的稀疏矩阵处理内核设计

3.1 概 述

为了充分挖掘 DFU 加速器上 SpMM 内核的性能，论文提出 DFU-SpMM，其由面向数据流指令复用的分层稀疏矩阵处理方法和面向流图优化的自适应汇编代码生成方法组成，完整工作流程如图 3 所示。

为了进行数据流图重用以及后续的分块负载均衡，DFU-SpMM 首先对矩阵中的稀疏行进行基于 2 的次幂非零元数目的数据重组织，以得到流图重用块以及流图重构块并分别对其进行压缩，对于不同的稀疏块采用不同的稀疏矩阵处理方法。进一步地，DFU-SpMM 将前述压缩后的稀疏块传入汇编代码生成模块，根据 DFU 加速器的计算资源和存储资源来

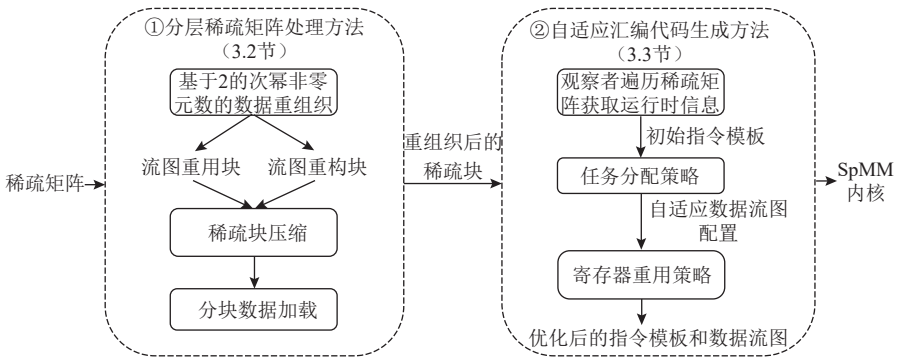


Fig. 3 The workflow of DFU-SpMM

图 3 DFU-SpMM 的工作流程

自适应地为不同的稀疏块生成汇编代码。

3.2 面向数据流指令复用的分层稀疏矩阵处理方法

现有工作为了优化稀疏矩阵的数据局部性,往往会采用基于相似度的重排序算法对不规则分布的稀疏数据进行聚类,基于相似度的 LSH 重排序算法是其中的代表性工作。尽管 LSH 算法能有效改善 GPU 上 SpMM 内核的缓存命中率,增加内核执行过程中的数据重用次数,但 DFU 上数据重用往往是通过 PE 之间的数据流动进行的,稀疏数据的重用可能会导致数据流图发生变化,进而引起 PE 阵列启动间的数据流图重传,此外,DFU 的数据流动是通过在不同 PE 之间的寄存器间传输数据实现的,由于 DFU 上寄存器数目的限制,难以维护寄存器资源用于数据流动,因此以相似度为衡量指标的重排序算法并不适用于 DFU 上的 SpMM 内核优化。同时由于 LSH 算法预处理开销往往很大,相较于 SpMM 内核的性能提升,重排序开销是不可忽略的。

基于上述观察,本节针对 DFU 数据流加速器数据流图重用的特性设计一种轻量级数据重组织算法 ReSparse,在较低的预处理时间开销下对稀疏矩阵进行规则化处理,以缓解 DFU 上 SpMM 内核频繁数据流图重传的问题,以达到数据流友好的目的。这里的规则化是指 SpMM 内核执行时需满足 3 个条件:1)加载的非零元数目相同;2)加载的输入密集数据数目相同;3)每次启动时发生数据流动的 PE 数目不变且 PE 之间的数据流动方向不变。

条件 1)和条件 2)确保了 SPM 缓存空间划分相同,这为访存指令中的 SPM 地址保持不变提供支持,进而保证流图节点不变,而条件 3)旨在确保流图节点间的数据流动恒定不变。

由于任意一个整数都可以分解为 2 的次幂的和,为了使加载的非零元数相同且处于同一稀疏行上(即发生数据流动的 PE 数目相同),ReSparse 将稀疏行划分为多个具有 2 的次幂个非零元的子行,并将划分后的子行聚类到多个稀疏块中,确保每个稀疏块内的子行都具有相同数目的非零元,进而使稀疏数据在 SPM 缓存中的排布满足条件 1)。对于条件 2),ReSparse 为每个非零元均加载一行输入密集数据进入 SPM 缓存(即使不同非零元可能需要相同的密集数据),这会带来一定的存储空间浪费,为了确保该策略能够带来收益,ReSparse 维护 1 个阈值来限制加载的冗余密集数据的总量。对于条件 3),由于稀疏块中的子行都具有相同的非零元数,因此处理子行所需的 PE 数目相同,若将稀疏数据以子行为粒度从 SPM 缓存加载到 PE 阵列进行计算,则阵列每次可处理的子行数目不变。在此基础上可将 PE 阵列划分为不同的区域,不同区域负责处理不同的子行。而单个子行内部仅有稀疏数据的数据重用,这确保每个子行内部的数据流动方向不变。

图 4 所示为 ReSparse 算法的核心思路:①根据非零元个数是否为 2 的次幂将稀疏行分别存储到 bucketA 和 bucketB 两组桶中进行粗粒度的聚类。桶组 bucketA 维护 1 组非零元个数为 2 的次幂的桶,分别存储非零元个数为 2 的次幂的稀疏行,代表可能重用数据流图的稀疏行;桶组 bucketB 维护 1 组非零元个数大于 2 的次幂的桶,分别存储非零元个数大于 2 的次幂的稀疏行,代表无法重用数据流图但可能通过数据重组织转换为可重用数据流图的稀疏行。②为了让更多的稀疏行具有相同数目的非零元,增加数据流图重用的次数,对桶组 bucketB 中的稀疏行进行划分和聚类。取出桶组 bucketB 中的行,将其划分为非零元数为 2 的次幂的子行与不为 2 的次幂的子行,并分别存入桶组 bucketA 和桶组 bucketB 中,直到桶组 bucketB 中存储的行均已处理完毕。③为了确保 SpMM 内核在处理桶组 bucketA 中所有的稀疏行时能够重用数据流图,接下来根据 SPM 缓存容量计算出单次 PE 阵列启动能够从桶组 bucketA 中加载的稀疏行数 *RowNum*,如果桶组 bucketA 中的稀疏行数不为 *RowNum* 的整数倍,则对桶组 bucketA 中的稀疏行进行划分并聚类;否则,继续处理桶 bucketB 中的稀疏行。循环此过程,直到将桶组 bucketB 中的稀疏行都处理完毕。

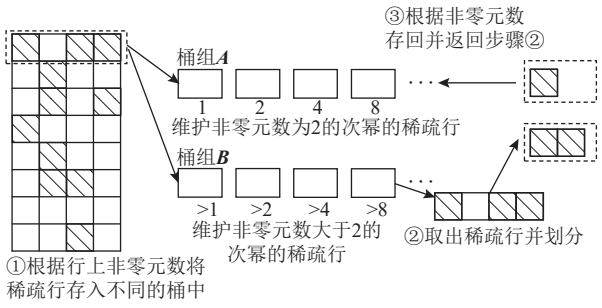


Fig. 4 Processing diagram of ReSparse

图 4 ReSparse 处理过程图

此外,为了防止稀疏行划分过程提前停止而导致未能充分挖掘数据重用块,该数据重组织方法在算法停止后还应综合已划分得到的规则稀疏行对内核计算过程的影响以及数据流图重传开销,对稀疏矩阵进行全局化的稀疏行划分以尽可能地增加内核执行时的数据流图重用率。

需要注意的是，在对稀疏行进行划分后会得到2个稀疏子行，若实时构造子行则会产生大量数据移动开销。为了尽可能少地产生时间开销，ReSparse在对稀疏行进行划分时并不会实际构造划分得到的稀疏子行，只会记录其在原来稀疏行中的位置信息并加入桶中。该算法只有在最后对稀疏矩阵和密集矩阵在存储空间进行重新排布时才会对原来的数据进行移动，这是由于DFU在从DRAM主存加载数据到SPM缓存时，需要确保数据处于连续的内存空间中。由于SpMM内核的间接访存的特性，无论是否使用ReSparse对数据进行重组织，都需要重新在DRAM中对密集矩阵数据进行重排布，这一时间开销是不可避免的。

尽管数据流图重用能够有效减少数据流图重传次数，但也带来了稀疏数据对应的输出密集矩阵数据需要重复加载到SPM缓存的问题，造成SPM缓存资源的浪费，潜在增加数据传输的次数，若额外的数据传输开销大于数据流图重用产生的收益，可能会降低DFU上SpMM内核的性能。

为了解决这一问题，这里将划分得到的稀疏行数目记为α，原本就在桶中存在的稀疏行数目记为β，数据传输时间记为μ，流图传输时间记为γ，当$(\alpha+\beta)/\beta\times\mu>\gamma$时，则停止ReSparse算法的执行，否则继续对桶中稀疏行进行划分和聚类。具体而言，当额外数据传输时间大于流图传输时间时，说明流图重用已无法弥补额外数据传输带来的时间开销；否则，说明流图重用节省的时间可以弥补冗余加载带来的数据传输开销。

为了高效处理ReSparse算法重组织得到的稀疏矩阵，提出分层次的稀疏矩阵处理策略，如图5所示，核心思路是将重组织后的稀疏矩阵按照行上非零元个数划分为多个规则的数据流图重用块(dataflow graph reuse block)和一个不规则的数据流图重构块(dataflow graph reconstruction block)，对不同的稀疏块使用不同的内核处理策略。数据流图重用块和数据流图重构块具有不同的特性，数据流图重用块中存储的是ReSparse重组织后的规则稀疏子行的集合，1个数据流图重用块根据SPM缓存容量可以划分为多个子块(SubBlock)，这些子块是一次可加载进SPM缓存的稀疏行的集合，子块间可以重用数据流图。而数据流图重构块中存储的仍是不规则的稀疏行，无法重用数据流图，每次PE阵列启动需要加载其中的一部分非零元并重新传输数据流图。尽管数据流图重用块能够对数据流图进行重用，但需要加载重复的密集矩阵数据，造成SPM缓存资源的浪费。而数据流图重构块则不需要对重复的密集矩阵数据进行加载，充分利用有限的SPM缓存，但其会带来额外的数据流图重传开销。

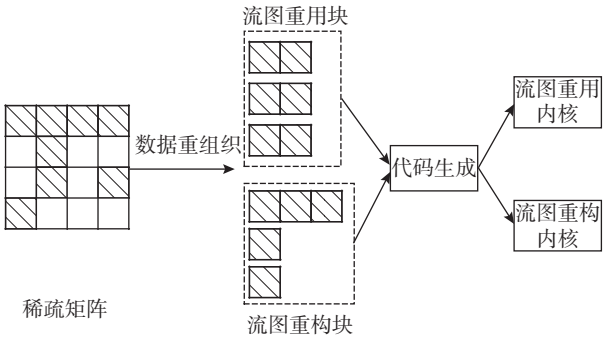


Fig. 5 Hierarchical sparse matrix processing strategy

图5 分层稀疏矩阵处理策略

因此，对于数据流图重用块和数据流图重构块需分别使用不同的处理方式。具体而言，对于规则的数据流图重用块，为了避免PE阵列跨行处理稀疏数据时对中间结果归约产生同步开销以及潜在的数据流图重构的问题，论文采用按行分配的原则对其进行处理，每个PE负责处理相同行数的非零元。同时由于流图重用块内的稀疏行上非零元数目相同，不会产生因负载不均衡引起的PE计算资源空闲的问题。而对于不规则的数据流图重构块中的稀疏数据，由于其内部稀疏行上非零元数目不同且其不具有规则的计算模式，每次启动都需要重传数据流图，因此为了确保PE阵列上的计算资源间负载均衡，采用按非零元数进行分配的方式对其进行处理，需要注意的是，采用按非零元数分配的方式可能会导致PE之间需要进行归约，因此，在处理流图重构块时需要根据稀疏数据的分布构造不同的数据流图，以确保结果的正确性。

3.3 面向流图优化的自适应汇编代码生成方法

DFU上SpMM汇编指令自动生成需要解决2个问题：1)稀疏数据分布不规则，这导致无法直接推出其在SPM缓存空间中的具体地址，如何获取数据在SPM缓存中的地址信息并判断应该为每个PE生成哪些汇编指令；2)PE阵列上多个PE可以并行执行，如何分配数据给PE来充分利用DFU的并行处理能力同时提高数据重用。

上述2个问题与处理稀疏矩阵的方式密切相关，为此需要先确定生成的SpMM内核处理稀疏矩阵的方式。SpMM内核处理稀疏矩阵的方式主要分为3种：内积(inner product)、外积(outer product)和逐行

遍历(row-based),如图 6 所示。

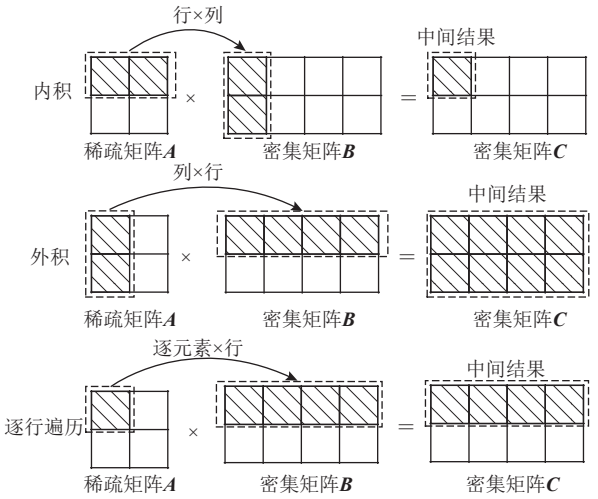


Fig. 6 Three sparse matrix processing methods of SpMM kernel

图 6 SpMM 内核的 3 种稀疏矩阵处理方式

图 6 中,内积是指每次计算稀疏矩阵的 1 行和对应的密集矩阵列的乘积进而产生的 1 个结果密集数据;而外积是指计算稀疏矩阵的 1 列和对应的密集矩阵行的乘积进而产生的整个中间结果矩阵的密集数据;逐行遍历的方式是指通过单个稀疏数据与对应的密集矩阵行相乘而获得的 1 行输出密集数据的中间结果。

由于内积的方式访问稀疏数据需要在运行时根据稀疏数据的列号访问密集矩阵的行号,这一操作导致访问的密集矩阵数据不连续,而 DFU 是通过 SIMD32 的形式访问连续数据,因此内积会导致无法充分利用 DFU 的内存带宽。此外,外积的方式则会导致产生的中间矩阵占用大量的存储空间,而片上的寄存器和 SPM 缓存有限,因此可能会使加载到 SPM 缓存中的非零元以及单次启动可运行的 PE 计算资源的数目受限,进而造成 PE 计算资源空闲并增加数据传输次数,产生更多的数据传输时间。而 Row-based 的方式可以连续访问密集矩阵数据,充分利用 SIMD32 提供的内存带宽,并且产生的中间结果相较于外积的方式更小,因此所设计的 SpMM 内核采用行遍历的方式访问并处理稀疏数据。

ReSparse 数据重组织后,稀疏矩阵数据被划分成了数据流图重用块以及数据流图重构块,它们具有不同的特性,采用观察者-执行者的思想分别对其处理。观察者的作用是遍历稀疏矩阵,获取稀疏数据的位置信息;执行者的作用是将生成的汇编指令和数据流图部署在 PE 阵列上执行。

对于数据流图重用块,由于稀疏数据在 SPM 缓存中顺序存储,且每个非零元对应 1 行输入密集数据,因此,在使用观察者遍历稀疏矩阵时,根据每行非零元数目可以得到本次启动时加载的密集数据的行数,进而直接根据行数确定内核涉及的矩阵的基地址。

对于数据流图重构块,为了尽可能地提高内核计算时的性能,若稀疏数据发生重用,不需要和数据流图重用块一样将其重用的密集数据重复加载到 SPM 缓存中,因此,重构块处理的密集数据在 SPM 缓存中不是规则存储的,无法通过稀疏行的行长信息直接推断出各矩阵的基地址。为了获取具体地址信息,需要在生成汇编指令前通过观察者预遍历稀疏数据来确定本次 PE 阵列启动将加载多少非零元,以尽可能填满 SPM 缓存,避免不必要的缓存浪费,进而确定稀疏矩阵 $\boldsymbol{A}$、密集矩阵 $\boldsymbol{B}$ 和密集矩阵 $\boldsymbol{C}$ 在 SPM 缓存中对应的基地址。

此外,为了尽可能提高观察者生成的内核的性能,从任务分配和寄存器重用 2 个角度设计优化策略指导代码生成过程。

1)基于任务分配的数据流图优化策略。SpMM 内核计算主要涉及到 3 个矩阵维度,这里定义输入稀疏矩阵 $\boldsymbol{A}$ 的数据规模为 $M\times N$,输入密集矩阵 $\boldsymbol{B}$ 的数据规模为 $N\times K$,输出密集矩阵 $\boldsymbol{C}$ 的数据规模为 $M\times K$,为了充分利用 DFU 加速器的并行能力,需要将 16 个 PE 计算资源分配到 M 维、N 维和 K 维上,因此为 PE 计算资源分配任务本质上是对 3 个不同维度上的非零元进行划分,不同维度的划分方式导致 PE 之间具有不同的数据流动关系。

在行遍历的处理方式下,若 PE 计算资源在矩阵的 K 维并行,则不同 PE 将处理密集矩阵 $\boldsymbol{C}$ 同一行上不同的数据以及处理稀疏矩阵 $\boldsymbol{A}$ 中相同的数据,因此可以在它们之间配置数据流动关系来重用稀疏矩阵 $\boldsymbol{A}$ 的数据;若 PE 在矩阵的 N 维并行,由于它们处理稀疏矩阵 $\boldsymbol{A}$ 的同一行上不同的数据但需要处理密集矩阵 $\boldsymbol{C}$ 同一行上相同的数据,因此可以通过数据流动来重用密集矩阵 $\boldsymbol{C}$ 的数据。同时为了确保结果的正确性,不同的 PE 之间需要对数据进行归约,尽管归约操作也可以通过 PE 间的数据流动来完成,但也会引入潜在的同步开销;若 PE 之间在 M 维上并行,它们处理稀疏矩阵 $\boldsymbol{A}$ 同一列上不同的数据,但由于稀疏矩阵的不规则分布以及数据流动所需要的寄存器资源存在限制,导致它们很难进行数据重用,因此在这种情况下无数据流动关系。

通过上述分析可以得出，在 SpMM 内核计算过程中，K 维并行能够最大程度地利用 PE 之间的数据流动，而 M 维并行则没有数据流动。因此，在为 PE 分配任务时，优先在 K 维上进行并行划分，如果还剩下 PE 计算资源，则对 N 维进行划分。最后，为了防止 PE 计算资源空闲，再将 M 维分配给剩下的 PE。根据此原则，可以自适应地确定生成的汇编指令与 PE 之间的对应关系并将指令部署到对应 PE 上，进一步生成复制指令来配置 PE 之间的数据流动关系，充分利用 DFU 加速器数据流动的特性以减少对 SPM 缓存的访问次数。

2）基于寄存器重用的数据流图优化策略。PE 阵列上共有 64×32（2 048）个寄存器，在负载均衡且最大 Task 并行（最多划分 4 个 Task）的情况下，每个 PE 最多只能使用 32 个寄存器。PE 计算资源通过不同功能部件并发执行，因此在执行时寄存器中的数据会不断更新。如果 SpMM 内核执行中出现了对密集数据的重用则可以配置不同 PE 的寄存器间的数据流动来减少对 SPM 缓存的访问，但也会使得该数据在流出前不能被更新，进而产生同步开销，因此，难以通过输入密集矩阵的数据流动进行数据重用。

为了提高在行遍历方式下 SpMM 内核对输入密集矩阵的数据重用，本文考虑提高 PE 内部的寄存器重用。前面提到在最大并行的情况下，每个 PE 最多使用 32 个寄存器，而在前面的行遍历方式中（1 个稀疏数据与对应的 1 行输入密集数据相乘，得到对应的 1 行输出密集数据），PE 需要加载 3 个矩阵的相关数据，分别是输入稀疏矩阵 $\boldsymbol{A}$、输入密集矩阵 $\boldsymbol{B}$ 和输出密集矩阵 $\boldsymbol{C}$ 的数据，它们分别需要 1 个寄存器来存储。由于在行遍历的情况下，稀疏矩阵 $\boldsymbol{A}$ 的每个数据在进行 1 次计算后已完成其计算任务，因此只需重复使用 1 个寄存器，输出密集矩阵 $\boldsymbol{C}$ 的数据由于会一直保存在 PE 的寄存器中进行累加，因此也只需要 1 个寄存器。而输入密集矩阵 $\boldsymbol{B}$ 的数据在稀疏矩阵遍历过程中会不断变化，当再次访问列号与该数据行号相同的稀疏数据时，若寄存器中仍存在该数据，则能够重用寄存器中的数据，直接使用寄存器的数据进行计算；若寄存器中没有该数据，则需要重新从 SPM 缓存中加载，因此可以将剩下的 30 个寄存器用来存储矩阵 $\boldsymbol{B}$ 的数据来提高内核计算过程中输入密集矩阵 $\boldsymbol{B}$ 中数据的寄存器重用率。

这里将寄存器编号为 0~31，为了方便管理，在代码生成时将 0 号寄存器用来存储稀疏矩阵 $\boldsymbol{A}$ 的数据，31 号寄存器用来存储输出密集矩阵 $\boldsymbol{C}$ 的数据，1~30 号寄存器存储输入密集矩阵 $\boldsymbol{B}$ 的数据。在遍历稀疏数据生成汇编代码时，顺序更新存储输入密集矩阵 $\boldsymbol{B}$ 数据的寄存器，并维护 1 个哈希表来管理各密集数据所存放的寄存器的信息。访问密集矩阵 $\boldsymbol{B}$ 的数据时通过哈希表来查找上一次访问时其存储的寄存器号，并根据当前寄存器是否仍保存着该数据来生成汇编指令，如果数据仍在寄存器中，则直接生成计算指令而不是加载指令，直接进行计算；如果不在寄存器中，则将其存放在当前正更新的寄存器中，根据其在 SPM 缓存中的具体地址生成加载指令并配置加载指令与计算指令的数据流动关系。

3.4 汇编代码样例

生成的汇编指令以指令模板的形式表示，每个指令模板会被部署到对应的逻辑节点上；不同逻辑节点之间还需要根据数据依赖关系构造数据流动关系。为便于依赖关系的构建，本文将生成的指令模板进一步按指令类别和功能拆分为不同的指令块，并构造指令块之间的依赖关系。以 LD 指令块为例，其指令模板可表示为“LD, dstReg, offset, baseId”。其中，dstReg 表示加载目标寄存器，offset 表示地址偏移，baseId 表示基址编号。例如，当目标寄存器为 REG1、地址偏移为 197554176、基址编号为 0 时，对应的汇编指令模板为“LD, REG1, 197554176, 0”。

4 实验评估

4.1 实验软硬件环境

为了验证论文提出的稀疏矩阵高效处理内核 DFU-SpMM 数据流友好的特性，在硬件环境方面，实验在 Linux 服务器上进行，CPU 型号为 Intel® Xeon® Platinum 8357B，主机内存容量为 512 GB，且搭载了显存容量为 24 GB 的 NVIDIA GeForce RTX 4090 GPU 以及内存容量为 16 GB 的 DFU 加速器，更详细的硬件环境配置信息如表 1 所示。

Table 1 Experimental Hardware Environment

表 1 实验硬件环境

服务器型号	CPU 型号	主机内存容量/GB	GPU 型号	GPU 显存容量/GB
Supermicro Super Server	Intel® Xeon® Platinum 8357B	512	NVIDIA GeForce RTX 4090	24

本文对内核在不同处理器上的能效比进行对比分析，表 2 列出了 DFU 加速器与 RTX 4090 GPU 的峰值算力、内存带宽以及最大功耗等关键硬件参数信

息。需要注意的是,由于GPU是多个流多处理器(streaming multiprocessor, SM)并行执行内核计算,因此GPU共享内存展示的是所有SM使用的总容量。2种加速器的内存在使用方式上有部分差异,后续在实验分析部分进行区分。

Table 2 Key Hardware Parameters of DFU Accelerator and NVIDIA RTX 4090

表 2 DFU 加速器与 NVIDIA RTX 4090 的关键硬件参数

硬件	峰值算力/TFLOPS	内存带宽/GBps	共享内存/MB	全局内存/GB	Tensor Core 算力/TFLOPS	最大功耗/W
DFU 数据流加速器	1.2	51.2	6	16		30
NVIDIA RTX 4090	82.6	1 008	12	24	330	450

在软件环境方面,服务器的操作系统为Ubuntu 22.04.2 LTS,其Linux内核为5.15.0-78-generic。本文提出的DFU-SpMM内核主要基于C语言以及DFU加速器的汇编指令实现,在此基础上使用Python实现内核的数据重组织和代码生成方法,其中Python版本为3.9.21。为了与GPU上的先进工作进行对比分析,还在NVIDIA公司提供的通用并行计算平台CUDA上进行相关实验,实验测试所使用的主要软件环境配置信息如表3所示。

Table 3 Experimental Software Environment

表 3 实验软件环境

软件	版本号
操作系统	Ubuntu 22.04.2 LTS
Linux 内核	Linux 5.15.0-78-generic
CUDA	12.4
Python	3.9.21
Nsight Compute	2023.1.0
Pytorch	2.5.1+cu121
Numpy	1.26.4
Scipy	1.13.1

4.2 测试数据与实验方案

4.2.1 实验数据集

实验选取6种现实世界中用于节点分类任务的开源数据集。其中,Cora, Citeseer, Pubmed数据集也为不同领域文献引用分析的学术数据集,它们的特点是均具有较大的特征维度,能够充分利用GPU和DFU加速器的内存带宽。其中Pubmed数据集具有相较另外2个数据集更大的数据规模,为内核提供了更多的并行机会,因此,这些数据集能够从多方面评估稀疏矩阵处理内核的性能。

然而上述数据集的数据规模均比较小,为了评估内核在高负载情况下的性能,实验还选取DD, YeastH, Protein等中大型数据集进行实验,它们均为蛋白质相互作用研究的数据集,这些数据集提供更多的节点以及更复杂的节点间的连接关系同时也具有更低的特征维度,这使SpMM内核执行过程中的访存效率和负载均衡均面临挑战。实验选取的数据集详细信息如表4所示。

Table 4 Details of Datasets

表 4 数据集详细信息

数据集	节点数	边数	特征维度
Cora	2 708	5 429	1 433
Citeseer	3 327	4 732	3 703
Pubmed	19 717	88 676	500
DD	334 925	1 686 092	89
YeastH	3 138 114	6 487 230	75
Protein	132 534	79 255 038	8

4.2.2 实验评估指标

如表2所示,DFU加速器的FP32理论峰值算力为1.2 TFLOPS,内存带宽为51.2 GBps,最大功耗为30 W;而RTX 4090 GPU的FP32理论峰值算力为82.6 TFLOPS,内存带宽为1 008 GBps,最大功耗为450 W,更重要的是其还具有更高性能的计算核心Tensor Core,理论峰值算力高达330 TFLOPS。而对于内存资源,DFU在运行时无法访问全局内存,因此其共享内存在功能上等效于GPU的全局内存,而DFU的共享内存(SPM缓存)容量仅为6 MB,GPU的全局内存却达到24 GB。

可以看到DFU数据流加速器无论是峰值算力、内存带宽还是内存资源都远比实验选用的NVIDIA RTX 4090 GPU要少,而DFU的优势在于具有更小的最大功耗。为了合理地评估DFU加速器与GPU上SpMM内核的性能,实验使用能效比和计算资源利用率这2种指标来对2种加速器上的工作进行比较。

能效比是算力与功耗的比值,具体公式为:能效比=实际吞吐量/运行功耗。通过上述参数可以粗略地算出,DFU数据流加速器使用FP32精度的理论能效比与NVIDIA RTX 4090 GPU使用FP 32精度的理论能效比分别为0.04 TFLOPS/W和0.183 6 TFLOPS/W,可以看出相较于峰值算力和内存带宽,能效比上的差距要小很多。但这只是理论情况,实际情况中能效比会根据硬件的工作负载动态变化,为此将能效比作为指标是非常有意义的。计算资源的利用率表

示计算单元在进行 SpMM 内核计算的时间里的实际工作时间，这能够看出内核对硬件资源的利用程度，为此，实验将计算资源利用率作为第 2 个评估指标。

4.3 实验结果与分析

4.3.1 SpMM 内核综合性能评估

1）能效比评估。图 7 所示为 RTX 4090 GPU 环境下的 NVIDIA 稀疏运算库 cuSparse 以及现有先进工作 HP-SpMM 和 DTC-SpMM 这 3 种 GPU 上 SpMM 内核与 DFU 加速器上的 DFU-SpMM 内核在 6 种经典数据集上的能效比差异，为了直观展示能效比的提升效果，实验数据进行了归一化处理，并以 cuSparse 为基线来观察各内核相较于 cuSparse 的能效比提升效果。同时考虑到 DFU 硬件架构与 GPU 不同，为了验证本文中流图重用方法的有效性以及 DFU-SpMM 对数据流加速器的友好性，实验将 DFU 上未进行数据流图重用的内核 DFU-base 也纳入对比范围。

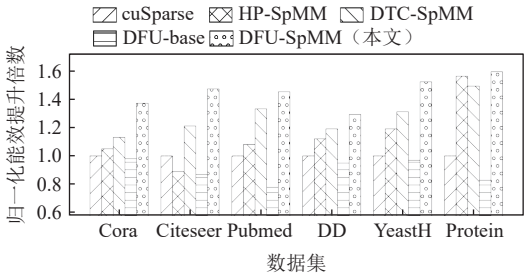


Fig. 7 Energy efficiency ratio of SpMM kernel in different hardware environments

图 7 不同硬件环境下的 SpMM 内核能效比

实验结果显示，DFU-SpMM 内核相较于 RTX 4090 环境下的基准 cuSparse 达到平均 1.38 的能效比提升，而对于 DTC-SpMM 和 HP-SpMM 分别实现平均 1.09 和 1.23 的能效比提升。实验测试过程中，DFU-SpMM 内核的功耗通常仅 1W 左右，而 GPU 上的内核在处理小数据集（如 Cora, Citeseer 等）和大数据集（如 YeastH）时的功耗分别达到 70 W 与 140 W 左右，这是由数据流架构加速器数据驱动的特性导致的，其只会在数据到达时激活计算单元并产生功耗且没有控制流架构中的分支预测等指令开销，因此尽管在 4.2 节中经过粗略分析得到 2 种加速器的理论能效比相差不大，但内核实际执行过程中 DFU 加速器具有更大的能耗优势。而 DFU-SpMM 内核相较同样在 DFU 加速器上的 DFU-base 内核实现 1.51 的能效比提升，这验证了提出的稀疏矩阵处理内核的有效性，其主要原因是 DFU 加速器上 SpMM 内核执行过程中产生了大量的数据流图传输开销，而 DFU-SpMM 内核的数据重组织方法则在一定程度上消除了其带来的影响。

对于 Protein 数据集，DFU-SpMM 表现不好的原因主要是该数据集的特征维度很小（仅为 8），而 DFU 通过 SIMD32 指令 1 次访问 32 个数据，因此在该数据集上 DFU-SpMM 无法充分利用访存带宽。尽管在该维度下，代码生成过程的任务分配策略会为特征维度上的数据配置计算资源间的数据流动，但其也伴随着对中间结果的归约开销，这实际上也增加了无效计算的次数。而 DTC-SpMM 和 HP-SpMM 在该数据集上能效高的原因可能是多方面的：一方面，实验测试中利用 Nsight Compute（NCU）内核性能分析工具对这 2 种内核进行分析得到在该数据集中它们的计算访存比均非常高，这可能是由于 Protein 数据集的行上非零元数相较其他数据集更多且稀疏行上非零元数分布更均匀，通过 DTC-SpMM 和 HP-SpMM 的基于相似度的重排序算法能够有效提高 GPU 的缓存命中率，减少对低效内存的访问；另一方面，对于 DTC-SpMM，其压缩稀疏数据的方式在该数据集上能够获得更密集的稀疏块，提高 Tensor Core 的利用率，而对于 HP-SpMM，数据集的非零元均匀分布也能够更好地发挥其动态负载均衡方法，进而减少 GPU 上 SM 的空闲时间。

2）计算资源利用率评估。由于 SpMM 内核在 GPU 上的吞吐量通常为几百 GFLOPS 到几千 GFLOPS（即 TFLOPS），相较于 GPU 的理论算力差距很大，因此本部分实验利用 NVIDIA 的 NCU 内核级性能分析工具来测试并分析现有 SpMM 内核优化工作在 GPU 上的计算资源利用率，具体测试指标选取 GPU 上的 SM 的实际吞吐量占其可持续峰值吞吐量的百分比，该指标可以有效反映内核对 SM 的利用程度。而 DFU 加速器的 PE 阵列实际上是对标 GPU 上的一个 SM，因此对于 DFU-SpMM，使用 DFU 加速器执行内核时统计并反馈的 PE 阵列的计算资源利用率作为评估指标。

图 8 所示为 RTX 4090 GPU 环境下的 3 种 SpMM 内核以及 DFU 加速器上的 DFU-SpMM 内核处理不同数据集时的计算资源利用率。实验结果显示，DFU-SpMM 在大部分数据集上相较于现有工作都具有更高的计算资源利用率，相较于 NVIDIA 的 cuSparse 库，DFU-SpMM 达到平均 1.43 倍的计算资源利用率提升，而对于 DTC-SpMM 和 HP-SpMM 分别实现平均 1.71 倍和 1.10 倍的计算资源利用率提升。在 DD 和 Protein 数据集上 DFU-SpMM 内核的计算资源利用率偏低的

原因可能是更低的数据流图重用导致的, DFU 加速器的数据传输由实际计算数据和数据流图构成, 这导致计算资源会花费更多时间等待传输结束并启动 PE 阵列。此外, 在 Protein 数据集上 DFU-SpMM 内核的寄存器重用率相对在其他数据集上更低, 进而产生了更多的加载指令。

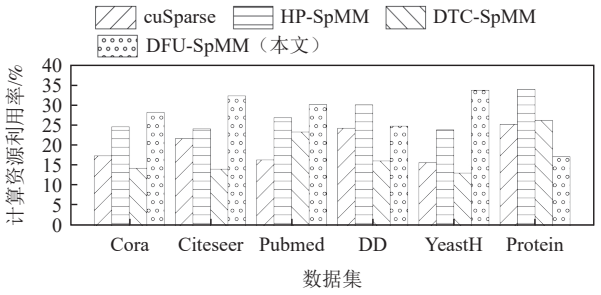


Fig. 8 Utilization rate of computing resources in different hardware environments

图 8 不同硬件环境下的计算资源利用率

实验测试过程中发现 DTC-SpMM 的吞吐量在大部分情况下均比 HP-SpMM 内核要高, 而 DTC-SpMM 内核的计算资源利用率的实际测试结果却偏低, 这主要与该计算资源利用率的指标有关。该实验的计算资源利用率中理论可持续峰值吞吐量是作为分母参与计算的, 而 Tensor Core 具有更高的理论峰值算力, 因此可能在 NCU 统计时是以相较 HP-SpMM 更高的峰值吞吐量计算的, 进而导致上述问题。需要注意的是, 尽管 NCU 默认将硬件的理论峰值算力设为可持续峰值吞吐量, 但实际测试数据显示 DTC-SpMM 内核在处理 Cora 数据集时的实际吞吐量为 741.34 GFLOPS(0.7 TFLOPS), 在此基础上推出的峰值算力约为 5.2 TFLOPS, 远低于硬件的理论峰值算力(330 TFLOPS)。尽管无法获取 NCU 内部具体的取值标准, 但相较于使用理论峰值算力作为分母来衡量利用率, 该指标更具有合理性。

4.3.2 内核处理策略评估

1)数据流图重用率评估。图 9 所示为 DFU 加速器上数据重组织前后的 SpMM 内核在处理 6 种数据集时数据流图重用率的相关实验数据, 其中 DFU-without 表示数据重组织前无法重用数据流图的内核, DFU-SpMM 表示进行数据重组织后生成的重用数据流图的内核, 进一步评估了数据重组织策略带来的数据流图重用收益。

实验结果显示, 稀疏矩阵数据重组织方法使 DFU 上生成的 SpMM 内核在处理 6 种稀疏数据集时的数据流图重用率占所有流图操作(流图重用和流图重传)的 63.42%, 有效减少数据流图重传带来的额外时间开销。值得注意的是, 由于 DFU-SpMM 内核是在 DFU 的编译器不支持稀疏计算编译的情况下设计的, 因此在使用数据重组织策略之前生成的内核 DFU-without 的数据流图重用率为 0, 在内核执行过程中没有数据流图重用。尽管 DFU-SpMM 内核已有不错的数据流图重用率, 但仍有一部分数据流图重传开销, 图中 DFU-SpMM 内核的数据流图重传一方面是稀疏行划分过程中数据重组织策略认为数据流图重用产生的冗余数据传输开销无法被数据流图重用带来的收益弥补进而停止稀疏矩阵的数据划分过程带来的, 停止划分后剩下的稀疏行会作为数据流图重构块的一部分处理, 进而产生数据流图重传开销。另一方面是由于不同数据流图重用块对应的数据流图不同, 因此内核在处理完一个数据流图重用块后需要为下一个数据流图重用块重新加载数据流图。

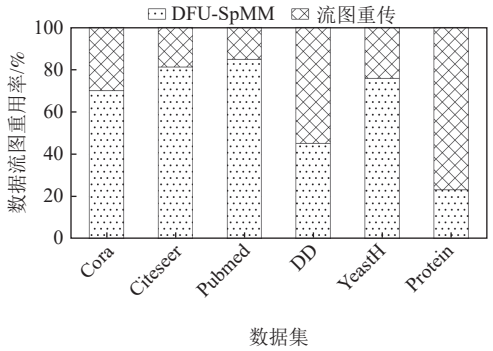


Fig. 9 Proportion of dataflow graph reuse and dataflow graph retransmission in different kernels before and after data reorganization

图 9 数据重组织前后不同内核中流图重用与流图重传占比

可以看到, DFU-SpMM 在 Protein 数据集上的流图重用率仅有 23%, 这是因为 DFU-SpMM 进行流图重用的部分为行上非零元数为 2 的次幂的稀疏行, 而 Protein 的稀疏行的行上非零元数分布比较均匀, 原本行上非零元数为 2 的次幂的稀疏行较少, 且为了权衡数据传输开销以及流图传输开销, DFU-SpMM 对稀疏行进行的操作更多的是补齐, 确保重组织后稀疏行的数目是单次内核处理最多能处理稀疏行的倍数。

2)内核性能提升效果评估。SpMM 内核的实际运算速度是衡量其高效性的重要指标, 但由于 DFU 提供的理论算力与 GPU 相差过大, 运算速度并不是 DFU 数据流加速器的优势, 因此本部分实验仅对 DFU 加速器上的内核进行实验对比。为了验证数据重组

织策略对 DFU 上 SpMM 内核性能的积极影响，实验在 DFU 加速器上，对比了数据重组织后生成的内核 DFU-SpMM 与未进行数据重组织的内核 DFU-without 在 6 种不同数据集上的执行性能，实验结果见图 10。

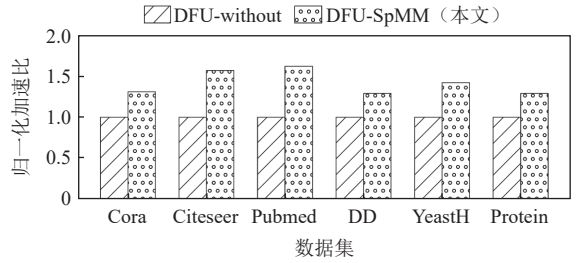


Fig. 10 Speedup effectiveness of SpMM kernel before and after data reorganization

图 10 数据重组织前后 SpMM 内核加速效果

实验结果显示，数据重组织策略为 DFU 上 SpMM 内核提供了稳定的性能提升效果，在 6 种不同的稀疏数据集上实现了平均 1.41 倍的性能提升。其中在 Pubmed 和 Citeseer 数据集上 DFU-SpMM 内核的加速比最高。观察发现，这 2 个数据集具有一定的共性，一方面，内核处理数据集时的寄存器重用率都比较低，如 DFU-SpMM 内核和 DFU-without 内核在处理 Pubmed 数据集时分别只有 4.73% 和 0.8% 的寄存器重用率，这导致寄存器重用策略带来的收益相较其他数据集更低。另一方面，Pubmed 和 Citeseer 数据集的初始规则行（即不进行重组织前已为 2 的次幂长度的稀疏行）占比分别为 73% 和 70%，因此数据重组织策略能够充分挖掘内核处理数据集时的数据流图重用，2 个方面因素叠加使得数据重组织策略能够带来更高的性能收益，此外，内核处理 DD 数据集时的加速比偏低可能也是这些因素导致的。

3）任务分配策略效果评估。由于数据流动的增加本质上会减少访存操作的数量，而在任务分配策略前内核无数据流动，本部分实验以计算访存比作为评估指标，对 DFU 加速器上进行任务分配但未配置数据流动的内核 DFU-without 以及配置数据流动的内核 DFU-SpMM 在 5 种不同数据集上进行实验测试和分析。此外，实验还在 RTX 4090 GPU 硬件环境下使用 NCU 分析 cuSparse 库在对应数据集上的计算访存比，以验证该策略在减少低效内存访问上的有效性。需要注意的是，由于 Protein 数据集能够有效提高输入密集数据在 GPU 多级缓存中的重用率，而该数据在 DFU 寄存器资源的限制下难以进行数据流动，因此本部分实验不将其纳入测试对比范围。

如图 11 所示，通过任务分配与数据流配置，DFU-SpMM 内核的计算访存比分别是未配置数据流的 DFU-without 内核与 cuSparse 的 3.11 倍和 1.09 倍，显著提升了 DFU 上 SpMM 运算的数据流动效率。DFU-SpMM 的数据流动主要由稀疏数据重用、输出密集矩阵重用以及归约操作的数据流动构成，其中，若任务分配导致不同 PE 处理相同稀疏行时会同时产生密集数据重用（不同 PE 处理同一稀疏行不同部分时对应同一输出密集行）和归约数据流动（对同一输出密集行的中间结果进行数据流动传输），因此在任务分配以特征维度优先的策略下，若特征维度相对较小会导致多个 PE 被分配在同一稀疏行，进而产生更多的数据流动，导致访存指令减少，因此 Pubmed 以及 YeastH 数据集更低的特征维度使 DFU-SpMM 内核具有相对另外 2 个数据集更高的计算访存比。此外，由于行上非零元数为 1 的稀疏行为 PE 阵列带来的数据流动收益与特征维度带来的一致，而 Pubmed 数据集的高特征维度减少其对数据流动的影响，因此计算访存比提升更显著。

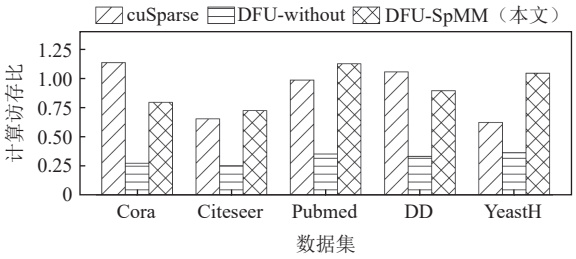


Fig. 11 Computational memory access ratio of SpMM kernel in different hardware environments

图 11 不同硬件环境下 SpMM 内核的计算访存比

此外，由于 cuSparse 库的内核是闭源的，难以通过代码分析其在各数据集上具有不同计算访存比表现的原因，本部分结合数据集和 GPU 上 SpMM 内核的通用优化技术对其进行分析。cuSparse 在 Citeseer 和 YeastH 数据集上的计算访存比显著低于其他数据集，而这 2 个数据集的共性为均具有大量非零元数为 1 的稀疏行，这导致很差的数据局部性。此外，GPU 的合并内存访问等高效访问模式需满足地址对齐且连续的条件，在这 2 个数据集上无法有效利用 GPU 的访存带宽，进而具有较低的计算访存比。

4）DFU-SpMM 各阶段时间评估。表 5 展示了 DFU-SpMM 执行过程中数据重组织、代码生成以及内核执行的时间开销。可以看到 DFU-SpMM 的数据重组织过程的时间开销还是可观的，因为其主要是对稀疏行进行拆分并进行分类，且数据重组织过程中数据搬运少，通过标记来记录新旧行的位置，并在最后

进行数据移动;且数据重组织更多的是对稀疏行的数目进行补全,以使其能够进行流图重用,因此开销较小。而代码生成过程的时间开销相对要大,特别是在 Protein 数据集该开销更显著,主要是因为 Protein 数据集中稀疏行的行长分布更加平均,进而导致流图重用率低,而这也意味着其不能很好地复用指令模板,需要生成很多不同的指令,增加了文件读写开销,进而增大了代码生成开销。与此同时,单次内核执行的开销相对较小,但是,在模型训练与推理过程中,稀疏结构往往是固定的,因此稀疏矩阵重组织与代码生成一般只需在首次执行时完成 1 次,后续可直接复用已生成的内核与重组织结果,从而将一次性预处理开销均摊到大量迭代中。以图神经网络训练为例,常见训练轮数为 100~400 个 epoch,且每个 epoch 至少会对每一层执行 1 次前向聚合(若计入反向传播则复用次数进一步翻倍);因此即便以 2 层模型计,单个算子在 1 次训练中的复用次数通常也可达到数百次甚至上千次,在这种复用机制下,预处理开销可以被有效摊销。

Table 5 Time Overhead of DFU-SpMM in Each Stage
表 5 DFU-SpMM 各阶段时间开销

数据集	重组织时间/s	代码生成时间/s	内核执行时间/ms
Cora	0.05	5.73	0.35
Citeseer	0.06	6.95	0.31
Pubmed	0.21	31.7	5.37
DD	4.48	241.10	121.62
YeastH	26.12	579.76	872.75
Protein	22.32	4 538.78	1 777.43

5 结 论

结合 DFU 加速器的特性,提出了一种面向数据流架构的稀疏矩阵高效处理内核以解决 DFU 数据流加速器无法高效处理稀疏矩阵运算的问题,通过数据重组织策略有效改善了 SpMM 内核的数据流图重用率,并提出了一种自适应汇编代码生成策略,解决 DFU 加速器不支持 SpMM 内核编译的问题并提高内核执行过程中的数据流动率。实验结果表明,相较于传统的 GPU 上的最新工作,本文方法能提供更高的能效比和计算资源利用率;且与原有的 DFU 系统相比,获得了全面的系统性能提升。本文的工作为在数据流加速器上实现高效的稀疏计算提供一种新的思路和解决方案。

作者贡献声明:蒋文斌负责总体设计,提出指导意见并参与论文修改;刘宝阳负责主要代码实现及论文撰写;董雨康协助完成方案整体设计与论文修改;沈欣海协助完成 DFU 相关代码的编写。

参 考 文 献

[1] Brown T, Mann B, Ryder N, et al. Language models are few-shot learners[C]//Proc of the 34th Annual Conf on Neural Information Processing Systems. New York: Curran Associates, Inc, 2020: 1877−1901

[2] Jawahar G, Sagot B, Seddah D. What does BERT learn about the structure of language?[C]//Proc of the 57th Annual Meeting of the Association for Computational Linguistics. Stroudsburg, PA: ACL, 2019: 3651−3657

[3] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[C]//Proc of the 31st Annual Conf on Neural Information Processing Systems. New York: Curran Associates, Inc, 2017: 5998−6008

[4] Zhang Biao, Haddow B, Birch A. Prompting large language model for machine translation: A case study[C]//Proc of the 40th Int Conf on Machine Learning. New York: PMLR, 2023: 41092–41110

[5] Yang Tao, Wu Rongyuan, Ren Peiran, et al. Pixel-aware stable diffusion for realistic image super-resolution and personalized stylization[C]//Proc of the 18th European Conf on Computer Vision. Berlin: Springer, 2024: 74−91

[6] Strubell E, Ganesh A, McCallum A. Energy and policy considerations for modern deep learning research[C]//Proc of the 34th AAAI Conf on Artificial Intelligence. Palo Alto, CA: AAAI, 2020: 13693−13696

[7] Hoefler T, Alistarh D, Ben-Nun T, et al. Sparsity in deep learning: Pruning and growth for efficient inference and training in neural networks[J]. Journal of Machine Learning Research, 2021, 22(241): 1−124

[8] Zaheer M, Guruganesh G, Dubey K A, et al. Big bird: Transformers for longer sequences[C]//Proc of the 34th Annual Conf on Neural Information Processing Systems. New York: Curran Associates, Inc, 2020: 17283−17297

[9] Wu Shiwen, Sun Fei, Zhang Wentao, et al. Graph neural networks in recommender systems: A survey[J]. ACM Computing Surveys, 2022, 55(5): 1−37

[10] Xia Tian, Ku W S. Geometric graph representation learning on protein structure prediction[C]//Proc of the 27th ACM SIGKDD Conf on Knowledge Discovery & Data Mining. New York: ACM, 2021: 1873−1883

[11] Li Ren, Cao Yanan, Zhu Qiannan, et al. How does knowledge graph embedding extrapolate to unseen data: A semantic evidence view[C]//Proc of the 36th AAAI Conf on Artificial Intelligence. Palo Alto, CA: AAAI, 2022: 5781−5791

[12] Huang Guyue, Dai Guohao, Wang Yu, et al. GE-SpMM: General-purpose sparse matrix-matrix multiplication on GPUs for graph neural networks[C]//Proc of the 32nd Int Conf for High Performance

Computing, Networking, Storage and Analysis. Piscataway, NJ: IEEE, 2020: 1−12

[13] Jouppi N P, Young C, Patil N, et al. In-datacenter performance analysis of a tensor processing unit[C]//Proc of the 44th Annual Int Symp on Computer Architecture. New York: ACM, 2017: 1−12

[14] Sarkar R, Abi-Karam S, He Yuqi, et al. FlowGNN: A dataflow architecture for real-time workload-agnostic graph neural network inference[C]//Proc of the 29th IEEE Int Symp on High-Performance Computer Architecture . Piscataway, NJ: IEEE, 2023: 1099−1112

[15] Wang Hongyi, Zhong Kai, Zhang Haoyu, et al. DySpMM: From fix to dynamic for sparse matrix-matrix multiplication accelerators[C]//Proc of the 61st ACM/IEEE Design Automation Conf. New York: ACM, 2024: 1−6

[16] Wang Runze, Hu Ao, Zheng Long, et al. An efficient GCNs accelerator using 3D-stacked processing-in-memory architectures[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 43(5): 1360−1373

[17] Fan Zhihua, Li Wenming, Tang Shengzhong, et al. Improving utilization of dataflow architectures through software and hardware co-design[C]//Proc of the 29th Int European Conf on Parallel and Distributed Computing. Berlin: Springer, 2023: 245−259

[18] Fan Zhihua, Li Wenming, Wang Zhen, et al. Improving utilization of dataflow unit for multi-batch processing[J]. ACM Transactions on Architecture and Code Optimization, 2024, 21(1): 1−26

[19] Jiang Peng, Hong Changwan, Agrawal G. A novel data transformation and execution strategy for accelerating sparse matrix multiplication on GPUs[C]//Proc of the 25th ACM SIGPLAN Symp on Principles and Practice of Parallel Programming. New York: ACM, 2020: 376−388

[20] Fan Ruibo, Wang Wei, Chu Xiaowen. Fast sparse GPU kernels for accelerated training of graph neural networks[C]//Proc of the 37th IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2023: 501−511

[21] Xie Xiaoru, Peng Hongwu, Hasan A, et al. Accel-GCN: High-performance GPU accelerator design for graph convolution networks[C]//Proc of the 42nd IEEE/ACM Int Conf on Computer Aided Design. Piscataway, NJ: IEEE, 2023: 1−9

[22] Wang Yuke, Feng Boyuan, Wang Zheng, et al. TC-GNN: Bridging sparse GNN computation and dense tensor cores on GPUs[C]//Proc of the 32nd USENIX Annual Technical Conf. Berkeley, CA: USENIX Association, 2023: 149−164

[23] Fan Ruibo, Wang Wei, Chu Xiaowen. DTC-SpMM: Bridging the gap in accelerating general sparse matrix multiplication with tensor cores[C]//Proc of the 29th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2024: 253−267

[24] Ye Zihao, Lai Ruihang, Shao Junru, et al. SparseTIR: Composable abstractions for sparse compilation in deep learning[C]//Proc of the 28th ACM Int Conf on Architectural Support for Programming Languages and Operating Systems. New York: ACM, 2023: 660−678

[25] Wang Ziheng. SparseRT: Accelerating unstructured sparsity on GPUs for deep learning inference[C]//Proc of the 29th ACM Int Conf on Parallel Architectures and Compilation Techniques. New York: ACM, 2020: 31−42

[26] Qin E, Samajdar A, Kwon H, et al. SIGMA: A sparse and irregular GEMM accelerator with flexible interconnects for DNN training[C]//Proc of the 26th IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2020: 58−70

[27] Asgari B, Hadidi R, Krishna T, et al. ALRESCHA: A lightweight reconfigurable sparse-computation accelerator[C]//Proc of the 26th IEEE Int Symp on High Performance Computer Architecture. Piscataway, NJ: IEEE, 2020: 249−260

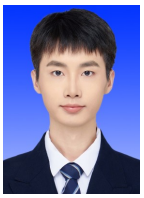
[28] Gao Yuan, Gong Lei, Wang Chao, et al. Algorithm/Hardware co-optimization for sparsity-aware SpMM acceleration of GNNs[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2023, 42(12): 4763−4776

[29] Kiningham K, Levis P, Ré C. GRIP: A graph neural network accelerator architecture[J]. IEEE Transactions on Computers, 2023, 72(4): 914−925



Jiang Wenbin, born in 1975. Professor. His main research interests include deep learning systems, parallel and distributed computing, and graph computing.

蒋文斌, 1975 年生。教授。主要研究方向为深度学习系统、并行与分布式计算、图计算。



Liu Baoyang, born in 2000. Master. His main research interest includes high performance computing.

刘宝阳, 2000 年生。硕士。主要研究方向为高性能计算。



Dong Yukang, born in 1995. PhD candidate. His main research interests include high performance computing and large language model inference optimization.

董雨康, 1995 年生。博士研究生。主要研究方向为高性能计算、大模型推理优化。



Shen Xinhai, born in 2001. PhD candidate. His main research interest includes large language model inference acceleration.

沈欣海, 2001 年生。博士研究生。主要研究方向为大模型推理加速。