SGCN：在深层图卷积网络加速器中利用压缩稀疏特征

Mingi Yoo[†, 1]、Jaeyong Song[†, 1]、Jounghoo Lee[†]、Namhyung Kim[‡]、Youngsok Kim[†, *]和 Jinho Lee[§, *]
[†]延世大学计算机科学系
[‡]三星电子[§]首尔大学电气与计算机工程系{skys7297, jaeyong.song, jounghoolee}@yonsei.ac.kr,
namhyungk11@gmail.com,youngsok@yonsei.ac.kr, leejinho@snu.ac.kr

arXiv:2301.10388v1 [cs.LG] 25 Jan 2023

摘要——图卷积网络（GCN）正日益流行，因为它们克服了以往神经网络适用范围有限的问题。GCN 的一个近期趋势是采用深层网络架构。与深度仅约为两到五层的传统 GCN 不同，现代 GCN 借助残差连接，现已包含数十层乃至数百层。通过研究这类深层 GCN，我们发现其一个重要特征：中间特征稀疏性极高。这揭示了一个此前尚未出现的新机会，使加速器能够在 GCN 执行过程中加以利用。

本文提出了*SGCN*，一种快速且高能效的图卷积网络加速器，能够充分利用现代图卷积网络中的稀疏中间特征。SGCN采用多种技术，相较于现有加速器显著提升了性能和能效。首先，SGCN采用一种适用于图卷积网络的特征压缩格式。我们着重减少片外存储器流量，因为它往往是图卷积网络执行的瓶颈。其次，我们提出了可无缝处理压缩特征格式的微架构。具体而言，我们修改了图卷积网络的聚合阶段，使其能够处理压缩特征，并设计了一种组合引擎，无需增加内存流量即可输出压缩特征。第三，为了在稀疏性变化时更好地处理局部性，SGCN采用了稀疏性感知协作。稀疏性感知协作形成一种具有多个复用窗口的模式，使缓存能够容纳不同大小的工作集，从而适应不同程度的稀疏性。通过全面评估，我们表明，与现有加速器相比，SGCN的几何平均加速比达到1.66×，能效提高了44.1%。

关键词—图卷积网络、稀疏性、压缩格式、加速器

I. 引言

图卷积网络（GCN）正逐渐成为一种
一种日益流行的深度神经网络（DNN）类型，因为它们能够处理高度不规则的数据 [37]。它们已成功应用于节点/边分类 [93]，分子结构分析 [16]，以及包括自然语言处理 [75] 或场景理解 [83]等经典深度神经网络应用中，并取得了高质量成果。

尽管图卷积网络属于深度神经网络的一种，但众所周知，与经典深度神经网络相比，它具有独特的计算特性，

[1]共同第一作者。*共同通讯作者。

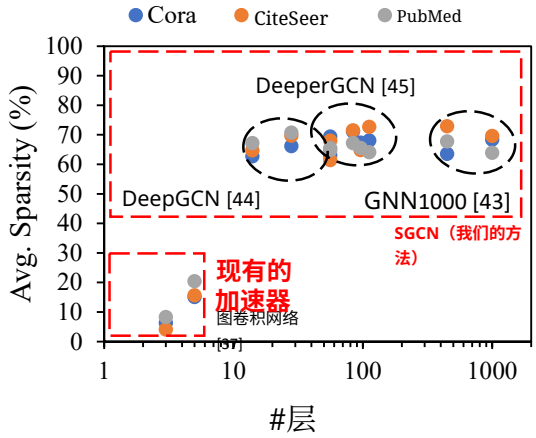


图1：传统和现代 GCN 中间特征的平均稀疏度（层数各异）。

因此需要专用的加速器架构
[80], [6], [46], [51], [17], [18], [88], [9], [47]。图卷积网络早期面临的挑战是如何处理输入图拓扑的高稀疏性。众所周知，图拓扑数据的稀疏性接近 100% [41]。受拓扑高稀疏性的推动，人们提出了多种方案，以利用执行过程中的混合特性 [80], [6]、应对负载不平衡 [51], [17], 、探索数据流 [47], [88],或重排拓扑 [18]。

与拓扑数据相比，现有 GCN 加速器通常将逐顶点特征数据视为稠密数组表示，因为特征的稀疏性远低于图拓扑的稀疏性 [17]。以特征访问为目标的先前工作通常侧重于输入特征 [17]，或者依赖于特征宽度朝输出层逐渐缩小这一事实 [17], [46]。这些方法适用于传统 GCN，尤其是在层数较少（≤ 5）且每层宽度经过精心调整的情况下。

然而，近期图卷积网络领域的发展带来了不同的情况。随着残差连接的引入，现代 GCN
[44], [45], [43], [10], [49] 如今拥有数十层、数百层，甚至上千层，而整个网络中的特征宽度始终保持不变。

值得庆幸的是，我们观察到现代 GCN 表现出更高的中间特征稀疏性，为提升吞吐量和能效提供了绝佳机会。如图所示，1，传统浅层图卷积网络的中间特征稀疏性一直很低（稀疏性最高达 20%）。然而，在具有残差连接的深层GCN中 [44]，[45], [43], 随着网络加深，中间特征稀疏性会急剧升高，最高可达 70%。充分利用这种中间特征稀疏性，可以提升性能和能效。考虑到大部分 GCN 执行都是内存密集型操作，并且会进行重复的随机访问，其预期收益十分显著。

然而，要充分利用特征的稀疏性，仍面临若干关键挑战。首先，我们需要一种适用于图卷积网络中间特征的特殊格式。直接将压缩稀疏行（CSR）等现有稀疏格式用于特征，可能会导致性能下降。尽管 CSR 被视为稀疏数据表示的事实标准，但对于我们观察到的稀疏度，以及由中间特征造成的动态稀疏度而言，它的效率并不高。

其次，现有的稀疏 DNN 加速器并不适合处理图卷积网络的稀疏特征。目前已有多种 DNN 加速器，通常针对卷积神经网络（CNN） [56], [33], [97]。然而，它们的设计目标通常是减少计算工作量。对于卷积神经网络而言，这种策略可能很有效，因为它们的计算密集度较高，减少计算次数（即乘累加运算）很容易转化为性能提升。然而，图卷积网络的计算复杂度相对较低，因为每个特征元素通常只涉及一次乘累加运算。在这种情况下，重点应放在减少内存流量上，而不是减少计算量。

最后，稀疏性水平的变化使得难以通过分块技术处理局部性。近期关于 GCN加速 的研究 [47], [88] 将图拓扑和特征矩阵进行分块，以缩小特征工作集，使其能够放入缓存。然而，采用这些技术时，动态稀疏度无法在静态时间点估算，因此难以确定合适的图块大小。

本文提出了SGCN，这是一种快速且节能的 GCN加速器，通过充分利用现代 GCN 的中间特征稀疏性，最大限度地减少片外 DRAM 访问。首先，我们提出了位图索引嵌入式原位 *CSR*（BEICSR）格式，专为 GCN执行量身打造。针对约 50% 的稀疏性，我们发现使用位图作为索引能够获得良好的压缩率。此外，将索引嵌入与内容相同的行中，可以显著提升访问的局部性。进一步地，BEICSR 执行原位压缩，因为这适合 GCN执行中的并行写入和随机读取。其次，我们提供了用于执行稀疏性感知GCN的微架构和流水线结构。我们设计了一个稀疏聚合单元，用于以压缩格式的特征执行 GCN 的聚合阶段。此外，我们还添加了一个

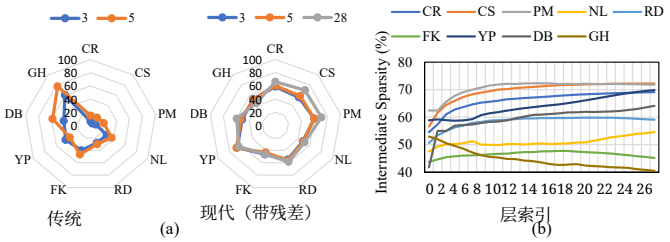


图2：（a）残差连接和层数对平均中间特征稀疏性的影响；（b）现代GCN各层的中间特征稀疏性。

原位压缩器，并将其与组合阶段的 ReLU单元结合，从而无需承担额外的内存访问开销即可完成压缩。第三，我们提出了稀疏性感知协作，以应对动态稀疏度导致的工作集大小变化。我们调整多个引擎，生成包含不同大小工作集的访问模式。因此，即使工作集大小超出预期，缓存也能捕获一个稍小、可容纳于缓存中的工作集，并复用部分特征。

总之，本文的贡献如下：

- 我们发现了利用现代深层GCN执行中间特征稀疏性的机会。
- 我们提出了BEICSR，这是一种专为GCN执行中的稀疏性感知聚合而设计的压缩格式。
- 我们提出了一种SGCN加速器微架构，可在GCN执行中利用稀疏特征数据。
- 我们提出了稀疏性感知协作，以应对动态稀疏性级别所导致的工作集大小变化。
- 我们对 SGCN 进行了全面评估，以展示其相较于先前工作的加速比和能效提升。

II. 动机

A. 中间特征稀疏性

在本节中，我们研究深层GCN的中间特征稀疏性。图2a展示了传统 GCN 的平均特征稀疏性 [37]以及现代深层 GCN [45] ，二者的区别在于是否存在残差连接。我们选择了 3、5 和 28 层，这些是传统（3、5 层）和现代（28 层）GCN 的常见配置。

如图所示1，少于五层的传统 GCN（图2a-Traditional）稀疏性较低，大致在 5%–30% 之间。请注意，28 层的传统 GCN 在所有数据集上都无法收敛。另一方面，使用现代深层 GCN（图 2a-Modern）时，我们可以观察到两点，这些特点与常规 GCN 不同。首先，添加残差连接会立即使稀疏性大幅提高 [66]。如图2a-Modern 所示，即使是浅层常规 GCN，添加残差连接也能使稀疏性超过 50%。当网络学习良好时，稀疏性约为 50% 是一种正常现象，而传统 GCN 的低稀疏性则意味着网络难以学习到有意义的特征集。其次，

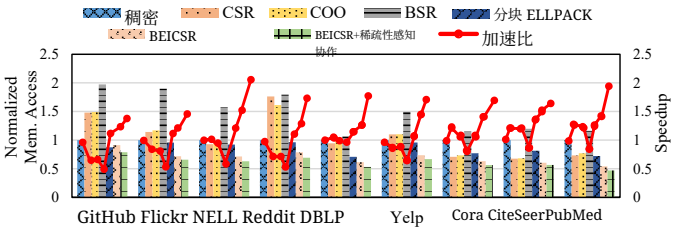


图 3：GCN 中间特征的各种稀疏数据表示对比。

总体而言，网络越深，稀疏性越高。尽管也有一些例外，图 2a-Modern 清楚地显示，较深的网络具有更高的稀疏性。在图 2b 中，我们绘制了从 '28 w/ residual' 网络测得的逐层稀疏性。结果表明，稀疏性因数据集而异，并且通常越接近输出层越稀疏。这与 [63], 的发现一致，可以解释为网络正试图寻找适用于分类任务的解耦表示 [25]。

通过这两项观察，我们发现，现代 GCN 在不同数据集上都存在较高的中间特征稀疏性，未来的 GCN 也很可能如此。SGCN 旨在利用这一特性，设计出性能和能效更优的加速器。

B. 新型加速器架构的必要性

本节将阐明新型加速器的必要性。目标是利用大量中间特征稀疏性带来的性能优势。图 3 中比较了分别采用稠密表示、压缩稀疏行、坐标格式、块压缩稀疏行、分块 ELLPACK 和所提出的稀疏格式（BEICSR）表示中间特征时，GCN 加速器的片外存储器流量和性能。我们使用在九个数据集上训练的 28层深层GCN；如第 II-A 节所述，这些模型的稀疏性约为 40% 至 70%。为便于观察，图中按平均稀疏性从低到高的顺序排列数据集。详细设置请参阅第 VI-A 节。

尽管中间特征具有巨大潜力，但结果表明，在 GCN 加速器上直接支持稀疏特征，带来的加速比很小，甚至会产生负加速比。彩色柱状条表示执行期间产生的片外存储访问次数，曲线则表示相应的加速比。以将特征视为稠密矩阵的基线"稠密"为参照，一种直接的方法是像处理稀疏图拓扑 [80], [17], [47], [18], [88] 或卷积神经网络中间特征 [57], [2]那样，为特征采用压缩稀疏行（CSR）数据结构。

由于 GCN执行众所周知地极度依赖内存，内存流量对性能至关重要。然而，CSR 每个非零值都需要额外的一个元素，在给定的稀疏性范围内无法减少内存流量。因此，朴素的 CSR 方法相较于基线只能获得较小的加速比，甚至会变慢。COO 格式的索引开销更大，因为它为每个非零元素都存储行索引和列索引。其他具有块稀疏性的稀疏格式也存在这个问题。块压缩行（BSR）和分块 ELLPACK 是常见的分块稀疏表示。不过，只有在空块很多时（例如，2×2），它们才有优势。由于 GCN 中间激活值很少呈现这种模式，因此这些格式并不适用于图卷积网络。

另一方面，标注为"BEICSR"的柱状条代表我们提出的方案，该方案使用 BEICSR 表示特征矩阵。针对观察到的稀疏程度设计的高效压缩格式，使我们的方案成功将稀疏性转化为内存访问次数的减少。此外，加入"稀疏性感知协作"（SAC）技术进一步提升了 BEICSR 的效率。这表明我们需要一种新的加速器，其加速比是现有方法无法实现的。

III. 背景

A. 图卷积网络

给定图拓扑和一组输入特征，传统图卷积网络（GCN）通过对输入特征执行各层的运算，生成输出特征。在基本的 GCN 中，第 l 层的输出特征 X^{l+1} 的计算方式如下：

$$X^{l+1} = \sigma(\tilde{A} \cdot X^{l} \cdot W^{l}) \tag{1}$$

其中，σ 是非线性激活函数（例如 ReLU），通常˜前面还会应用归一化函数。A 是图的邻接矩阵，X^l和W^l分别表示第l层的输入特征和权重。

过去，人们认为，超过五层的图卷积网络会导致准确率极低，这种现象通常称为过平滑 [82], [49]。因此，针对图卷积网络的设计工作大多着眼于谨慎配置少数几层的特征宽度，通常采用逐层缩减的方式。在这种情况下，一种颇有前景的方法是调整聚合与组合的顺序 [17], [47], [46]。由于改变这两个阶段的顺序不会影响式（1）的正确性，因此在缩减层中先进行组合，而在扩展层中先进行聚合，以减少计算量和内存访问次数 [46]。此外，有时对第一层进行特殊处理也会有所帮助 [17]。由于第一层的输入特征（即 X^1）并非计算得出，而是包含在数据集本身中，因此可能极为稀疏，尤其是在使用独热编码时。

然而，随着技术的发展，图卷积网络已开始采用残差连接，这改变了公式（1）为：

$$\begin{aligned} S^{l+1} &= S^{l+1}_{res} + S^{l}, \\ S^{l+1}_{res} &= \tilde{A} \cdot X^{l} \cdot W^{l}, \\ X^{l} &= \sigma(S^{l}). \end{aligned} \tag{2}$$

图 4 展示了这一操作。对 $\tilde{A} \cdot X^l$及其变体的计算通常称为聚合，该过程不涉及可训练权重。它是收集以下内容的过程：

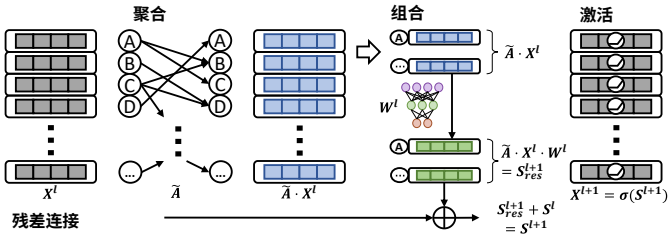


图4：现代GCN层的运算。

相邻顶点的特征并对其进行聚合。对 $X^l \cdot W^l$ 及其变体的计算通常称为组合，因为它通过线性组合输入特征和权重参数来计算下一层的特征。

引入残差连接后，人们发现GCN可以像卷积神经网络一样构建得很深，层数可达数十层 [10], [44], [45]，甚至上千层 [43]，并已应用于最先进的图卷积网络。随着网络层数加深，这些设计现在采用统一的特征宽度，因此阶段排序技术不再适用。此外，输入特征的稀疏性对总执行时间的影响已不大，因为加速比会被分摊到数十层上。幸运的是，这种残差连接的应用显著提高了中间特征的稀疏性，从而为提升GCN的性能带来了另一种机会。

B. 图卷积网络的硬件加速

通常，GCN 加速器的关键在于支持聚合阶段和组合阶段的混合特性。一些架构采用共享单元，通过可变数据通路支持这两个阶段 [17], [47], [51], ；另一些则采用独立单元，并将两个阶段进行流水线处理 [80], [9], [18], [88]。

表 I 总结了现有 GCN 加速器的特性。根据聚合阶段和组合阶段的执行顺序，GCN 加速器还可分为两类。聚合优先型加速器先执行聚合阶段，再执行组合阶段。也就是说，聚合优先型加速器先计算 $\tilde{A} \cdot X^l$，然后计算 $(\tilde{A} \cdot X^l) \cdot W^l$。另一方面，组合优先型加速器先计算 $X^l \cdot W^l$，然后计算 $\tilde{A} \cdot (X^l \cdot W^l)$。每种执行路径的优势取决于组合阶段后特征宽度是增大还是减小，一些方法建议利用这种差异 [47], [46]。然而，随着深层GCN [10], [44], [45], [43], 的出现，中间特征宽度通常在整个网络中保持不变，因此这种差异也就消失了。

图 5展示了基线图卷积网络加速器架构，与现有技术类似 [80], [9], [88], 其中绿色阴影标示的单元是为 SGCN 添加的。聚合单元使用 SIMD乘累加核心处理来自多个顶点的特征累加。拓扑矩阵

$\tilde{A}$ 假设采用 CSR 格式以利用高稀疏性。与图处理加速器类似 [21], [55], [40],，图读取器读取索引、顶点索引及相应的

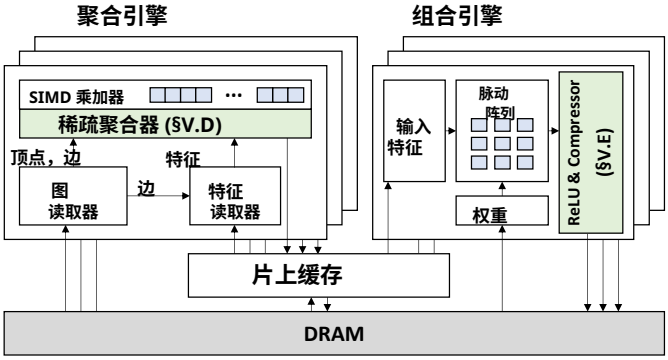


图 5：我们的 GCN加速器架构利用图的高稀疏性。绿色模块是为 SGCN 新增的。

边信息。根据边信息，特征读取器获取边目标顶点的特征向量。这些模块共同为 SIMD 核心供数，使其能够持续执行聚合而不会停顿。每个模块都有一个小型缓冲区，用于临时存储预取的值，以避免上游背压导致停顿。

由于对相邻顶点特征向量的访问在大范围数据中呈现高度随机的模式，因此使用大容量片上存储器作为全局缓存，其作用类似于现代 CPU 中的末级缓存。然而，工作集大小往往远超全局缓存的容量，导致无法有效利用特征向量访问中固有的局部性，使聚合阶段高度依赖内存。

组合引擎的核心是一个用于矩阵乘法的脉动阵列，与传统 DNN 加速器类似 [32], [38]。输入特征缓冲区和权重缓冲区分别向脉动阵列提供输入矩阵 X 和 W。输出会写回片外 DRAM，通常作为下一层的输入。

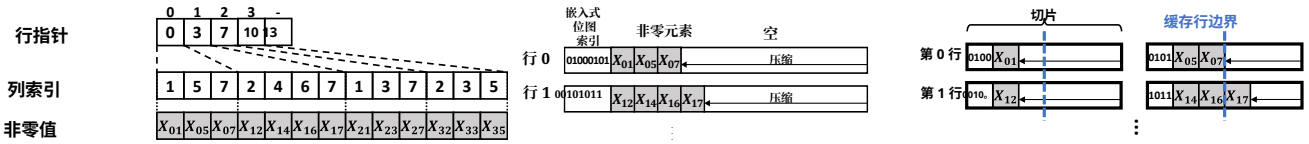
IV. 设计目标

基于上述动机，我们的目标是**为特征设计一种稀疏格式**，并**设计一种加速器微架构**，以高效处理这些特征。为实现这一目标，我们在推导 SGCN 方案时设定了以下设计目标。

首先，首要目标应是削减内存访问，而非计算或容量。首要

表 I：GCN 加速器比较

加速器	压缩特征?	目标层	残差	执行顺序
AWB-GCN [17]	✗	2	✗	组合优先
EnGN [51]	✗	2	✗	组合优先
HyGCN [80]	✗	1-2	✗	聚合优先
双向 [46]	✗	3	✗	Both
GCNAX [47]	✗	2	✗	Both
I-GCN [18]	✗	2	✗	组合优先
SGCN（我们的方法）	✓ **BEICSR (§V-A)**	**>5**	✓	**聚合优先**



(a) CSR 格式。 (b) BEICSR 格式。 (c) 支持特征分块的 BEICSR。

图6：特征矩阵的压缩格式。

众所周知，GCN执行的瓶颈在于聚合阶段，该阶段对内存的需求极高 [9], [18]。这表明，若不减少内存流量（例如通过零值跳过），仅优化计算量并非最佳选择。

其次，格式及其产生的访问模式应适用于缓存和DRAM。与卷积神经网络中可按顺序进行特征访问不同，GCN 聚合特征需要随机访问较小的逐顶点特征（即少量缓存行）。处理稀疏格式时，必须特别注意使每次访问都与缓存行和/或 DRAM突发长度对齐，以优化片外流量。此外，由于中间特征是上一层运行的结果，动态生成的，因此该格式应同时支持并行写入和读取。尤其是压缩导致行长可变时，访问模式很容易出现大量未对齐访问或存储体冲突访问。对于不支持多秩通道的现代存储器系统（例如 HBM），这类问题更为常见。因此，压缩格式及其执行方式应充分考虑存储子系统并加以利用。

第三，执行过程应兼容现有的图卷积网络加速器。例如，现有加速器会采用阶段间重叠 [80], [47] 或拓扑/特征分块 [47], [88]。直接应用压缩格式和执行流程可能会禁用这些技术，尤其是在无法静态估计可变稀疏度的情况下。采用稀疏性感知的图卷积网络执行时，必须兼容这些技术，才能实现最高性能。

V. SGCN架构

A. 压缩格式

SGCN压缩的最终目的是减少片外内存访问。这显然取决于我们为特征矩阵采用的压缩格式。与传统稀疏深度神经网络加速器相比 [56],[33], ，不同之处在于，由于拓扑矩阵A极度稀疏，图卷积网络的特征矩阵呈现出高度随机的模式˜。每次随机访问都会访问顶点特征数组的一部分，该部分通常跨越若干缓存行。

处理稀疏数据的一种天真方法是使用压缩稀疏行（CSR）。然而，直接使用天真的 CSR 并不能减少片外流量。采用 CSR 时，每个非零元素需要两个值：一个是列索引，另一个是元素值。因此，当稀疏性低于 50% 时，只会产生容量开销，而不会带来容量缩减，第二节对此进行了示例说明。。

为了解决上述问题，我们为中间图卷积网络特征提出了 BEICSR（位图索引嵌入式原位 CSR）格式。图 6 对比了同一组数据的传统 CSR 格式（图 6a）以及为 SGCN 提出的压缩格式（图 6b）。我们为该格式做出了以下设计选择：

嵌入式位图索引。我们不使用列索引，而是为每个顶点使用位图索引。例如，若一个大小为四的示例数组为 (0，0。3，0。5，0)，我们先放置 $0110'b$，后接非零数组 (0。3，0。5)。假设特征向量中的元素具有 50% 的稀疏性，且每个元素占用 32 位，我们便可计算位向量索引的开销。当特征向量宽度为 n 时，位图索引大小为 n 位，非零特征数据大小为 $32n \times 0.5 = 16n$。因此，在这种情况下，总索引开销仅为 6.25%（$n/16n$），远低于朴素 CSR 的开销；朴素 CSR 需要为每个非零元素存储一个整数索引。另一方面，如果对同样具有 50% 稀疏性的数据使用 CSR，列索引会带来 $32n \times 0.5$ 的开销，此外还需要行指针来指示数组的起始位置。加上非零特征数据的大小后，整体尺寸反而会增加，而非实现压缩。</output>

此外，我们提出将位图索引 嵌入存储非零值的同一数组开头。在压缩格式中，索引通常存放在独立的数组中。然而，这种选择会导致较差的内存访问模式。访问某一行的位向量索引时，包含其他行索引的周围缓存行总会在同一次内存访问中一并被访问。除非这些数据在被驱逐前得到重用，否则它们都被视为开销。观察访问模式可以发现，对位向量索引的访问几乎总是紧接着对非零值的访问。唯一的例外是位向量索引全为零时（即该行没有非零元素）。然而，考虑到稀疏性约为 50%，且每个元素之间的依赖性很小，这种情况不太可能发生。因此，将位图索引与非零值嵌入同一数组的做法能带来更好的内存访问模式。请注意，虽然有些格式使用位图索引 [74], [91], ，但它们依赖于完全为空的行 [74] 或块 [91] ，因此并不适用于 GCN 特征。

原位压缩。使用压缩格式提升性能时，常见的问题是长度不一。这不仅需要类似行指针的间接寻址数组，还会导致频繁的非对齐内存访问，从而产生流量开销。

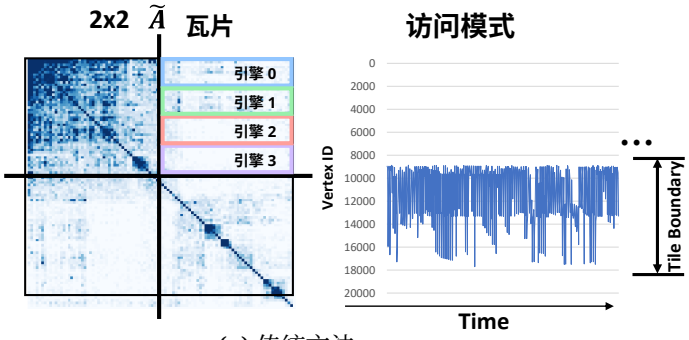
为了存储每行的偏移量，需要使用某种等同于行指针的间接寻址数组。相反，我们逐行压缩数据，并将每行存放在内存中预留的固定位置，就像该行未经压缩一样。尽管这无法提升内存容量，但我们认为，出于以下原因，这种做法几乎不可或缺。首先，它能减少片外流量，并使其与缓存行边界对齐。由于访问粒度仅涵盖少数几个缓存行，因此对齐尤为重要。每行预留的空间中有很大一部分是空的，因此不会从内存中读取。此外，每行的起始位置都可以按缓存行对齐，从而尽可能减少未对齐造成的损失。其次，这种做法允许在每层的输出端进行并行写入。直接采用变长存储格式需要串行处理，因为我们无法预先知道每个顶点生成的压缩特征数组的大小。这种做法会导致执行时间开销难以接受。最后，由于各行的大小一致，因此不需要间接寻址数组。只需将顶点 ID 乘以相应的步长，即可定位压缩数据，从而无需访问间接寻址数组。该方案减少了内存访问次数，有助于提高行缓冲区局部性，并提升内存带宽利用率。

B. 支持特征矩阵切片

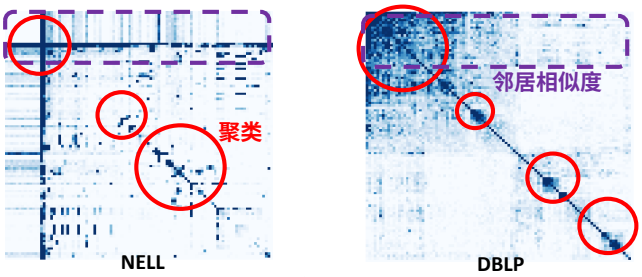
许多现代GCN加速器提出对特征矩阵进行切片，以提高数据复用率 [87], [88], [47]。将整行一并压缩时，访问特征矩阵的一个切片需要执行以下步骤：1）读取位向量索引，以查找与该切片对应的非零值范围；2）读取包含该切片非零值的多个缓存行；3）执行聚合。

上述访问流程的缺陷在于未对齐访问带来的开销。例如，当一个切片中有16个非零值（64字节）时，未对齐访问几乎总是需要访问两个64B缓存行，这抵消了稀疏性带来的优势。

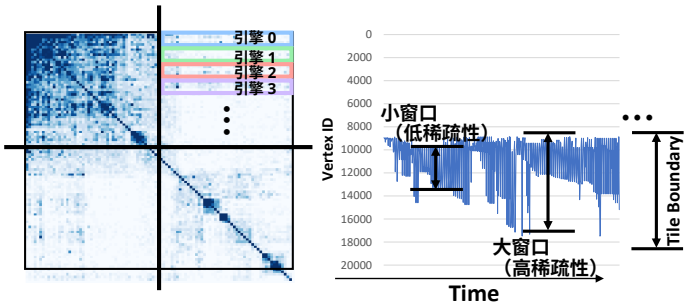
因此，我们采用切片式 *BEICSR*，如图6(c)所示。我们不再为整行设置一组位向量索引，而是对位向量进行分区，并将其嵌入各个对应单位切片的头部。随后，我们将切片对齐到突发边界。对每个切片采用第 V-A 节所述的原位压缩时，我们分配足以容纳其中最大非零值数量（即一个稠密切片）的内存空间。通过选择合适的单位切片大小C，可以最大限度地减少内存访问浪费，因为非零元素的数量方差较小，且离群值很少。需要更大的切片大小时，只需将多个单位切片合并为一个较大的逻辑切片；与一开始就使用大切片相比，这几乎不会带来任何差异。在 SGCN 的默认设计中，我们通过经验将 $C = 96$设为该值；对于单精度特征，这将占用 384 字节，稀疏性约为 50%



(a) 传统方法



（b）邻居相似度和聚类



(c) 稀疏性感知协作

图 7：（a）传统方法；（b）NELL 和 DBLP 图的邻接矩阵密度热图；（c）假设采用 2×2 分块的稀疏性感知协作。

连同嵌入式位图索引一起，可容纳在两到三个缓存行中。

C. 图拓扑分块的稀疏性感知协作

另一种有前景的 GCN 执行技术是图拓扑分块 [99], [47], [17]。将邻接矩阵划分为多个图块，有助于减小中间特征的工作集大小，使其能够适配缓存大小。现有方法通常通过离线分析来确定最优图块大小，常见做法是静态计算工作集大小 [88], [47]。然而，使用 SGCN 时，稀疏程度会随顶点、数据集和层的不同而动态变化。因此，很难估算最优图块大小；工作集大小一旦超过缓存容量，就会导致显著的性能下降。

图7a 展示了传统图分块在一个 2×2 分块示例中的工作方式。右侧显示了前 300 次读取所访问的特征对应的顶点 ID。当图被划分为图块时，工作集大小会被限制在

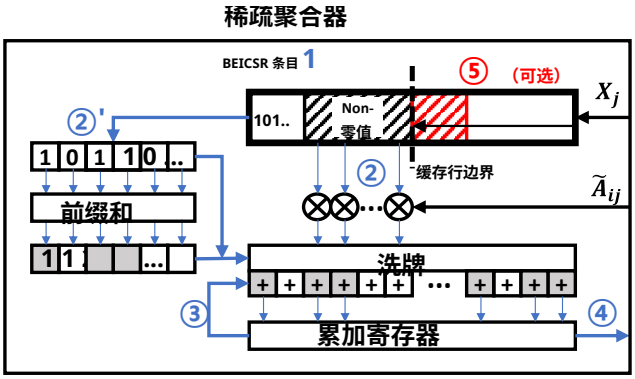


图 8：SGCN 稀疏聚合器单元微架构。

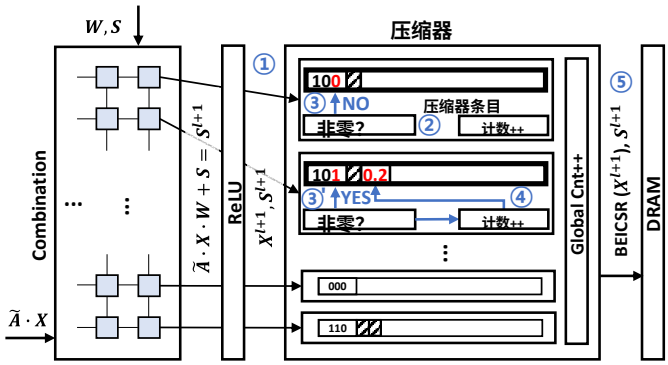


图 9：SGCN 压缩器单元微架构。

图块中所包含顶点的特征范围内，这些特征可以装入缓存。然而，当特征的稀疏性低于预期时，由于每个顶点包含更多非零值，有效工作集大小会增加。这存在超过缓存容量的风险，并且由于 [61], [95], [78]抖动模式，性能会迅速下降。

为解决这一问题，我们提出了稀疏性感知协作方法，通过改变访问模式，使缓存能够捕获大小各异的工作集。众所周知，图通常会形成社区簇 [19], ，相邻顶点之间存在邻居相似度 [7]。例如，图 [8] NELL 和 DBLP [72]中的示例密度图（见图 7b）表明，相邻行往往呈现相同的模式，并且在对角线附近具有明显的聚类特征。

稀疏性感知协作利用了这一特性，每个引擎以交错方式访问一小段顶点，如图 7c 所示。条带高度经实验确定为 32。由于邻居相似度和聚类，访问距离往往较短，并且能够捕获多个大小各异的工作集。因此，当稀疏性较高时，缓存会捕获较大的工作集窗口（称为“大窗口”）；当稀疏性较低时，缓存则可以捕获较小的工作集窗口（称为“小窗口”），以避免缓存抖动。

D. 稀疏聚合

图 8展示了 SGCN 中稀疏聚合器单元的微架构及其执行流程。SGCN 的每个稀疏聚合器都包含 16 个乘法器，可同时处理一个缓存行大小的数据。 1 选定特征矩阵 X 的一行后，会将其前 64 字节提取到聚合引擎中，其中头部包含位图，其余部分包含非零值。 2 非零值与 $\tilde{A}$ 中对应的边权重相乘，并广播给每个乘法器。 $2'$ 与此同时，并行前缀和单元会处理位图，将其中的 1 转换为指向非零值的反向索引。 3 位图和反向索引会发送到累加器。如果位图值为 1，则对应位置的累加器会加载乘法器的输出，并将其与当前值相加。 4 单个顶点的累加完成后，会将结果发送到组合引擎，执行组合、ReLU 和压缩。5 （可选）如果下一个缓存行中仍有非零值（由前缀和结果确定），则提取接下来的 64 位，再次执行 1 -4 。

E. 组合后压缩

图。9 展示了 SGCN 中的压缩器单元及其压缩过程。为避免压缩带来的额外内存访问，我们将压缩器置于采用输出驻留脉动阵列的组合引擎输出级。脉动阵列的每一行都分配一个压缩器条目，其中包含 BEICSR 缓冲区和压缩逻辑。1 组合阶段完成后，数据在经过残差加法和 ReLU 激活处理后，流入压缩器。2 压缩逻辑会检查输出值是否为零。3 如果输出值为零，压缩条目中的位图索引会追加一个‘0’。$3'$ 当输出值非零时，位图索引会累加一个‘1’，并且4 输出值会保存到计数器指向的位置。压缩器会持续对脉动阵列的每个输出值执行1- 4 所示的操作。5 压缩器处理完一个单位切片大小的数据后，缓冲区中存储的数据会被刷新到动态随机存取存储器，随后压缩器重新初始化。

F. 整合起来

图 10 展示了在基准架构（图 5）之上运行 SGCN 的整体流程。稀疏聚合单元接收 $\tilde{A}$，即 CSR 格式的图拓扑，以及 X^l，即来自上一层输出、采用 BEICSR 格式的特征。与 [80], [51], [88], 类似，我们采用基于行乘积的数据流，并对图拓扑 $\tilde{A}$ 和特征 X^l 进行分块，如 [51], [88],[47]所示。得到的 $\tilde{A} \cdot X^l$ 将是稠密的，因为$\tilde{A} \cdot X^l$的每一行都是由 X^l中的若干行加权求和得到的。

在稀疏聚合单元计算出一块 $\tilde{A} \cdot X^l$ 后，该块将被发送至基于脉动阵列的组合引擎。脉动阵列会将 $\tilde{A} \cdot X^l$与 W^l相乘，这本质上是一种通用矩阵乘法操作。要执行

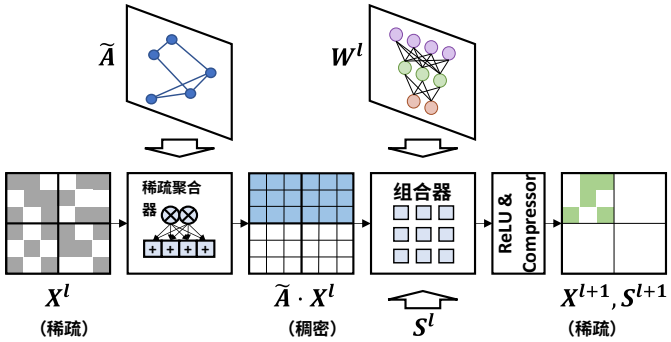


图10：GCN层的SGCN执行流程。

表 III：系统配置

加速器引擎	频率	1GHz
	组合	32×32 系统阵列
	聚合	16路 SIMD
#引擎	聚合	8
	组合	8
全局缓存	容量	512KB
	Ways	16
	替换	LRU
片外存储器	规格	HBM2
	峰值带宽	256 GB/s
	通道	8
	存储体	4×4

残差加法时，脉动阵列中的寄存器会初始化为 S^l 而非零。当$\tilde{A} \cdot X^l \cdot W^l$的一行切片计算完成后，会先经过 ReLU 激活，再送入压缩器单元。ReLU 激活的输出将作为下一层的输入特征 X^{l+1}。写入内存前，组合后压缩单元会将 X^{l+1} 的一行转换为 BEICSR 格式。在现代 GCN架构中，输入和输出特征宽度通常相同 [10], [43]。在这种情况下，X' 占用的容量与$\tilde{A} \cdot X$相同，因此压缩后的 X'会写入 $A \cdot X$原先所在的位置，以减少内存需求。

可选地，当输入特征（X^1）极其稀疏时，可改为在稀疏聚合器引擎上执行第一层的组合，以利用稀疏输入特征。尽管此技术仅适用于第一层，但它能显著提升某些数据集上的性能。与基线GCN加速器相比，SGCN无需增加太多实现成本。对于稀疏聚合器单元，只需额外添加前缀和单元来读取位图索引。对于SGCN压缩器单元，则在基本组合单元中添加全局计数器和非零比较器，以便在写入特征的同时写入位图索引。

VI. 评估

A. 实验设置

我们从多个来源选取了九个真实世界图数据集，如表 II 所示。由于实际训练模型的稀疏性对 SGCN 至关重要，我们根据 [44], [45] 中的实践，使用这些数据集训练了具有残差连接、每个顶点含 256 个特征的 28层GCN。

Cora (*CR*)、CiteSeer (*CS*)、PubMed (*PM*)、DBLP (*DB*) 是来自 [68], [72] 的引文网络图。每个节点（文档）都与其他节点相连，并具有由词袋构成的特征向量。NELL (*NL*) [8] 是来自永无止境语言学习项目的知识图谱。每个节点都有一个长度为 61,278 的独热向量。Reddit (*RD*) [81] 是常用于 GCN 内核评估的图。Flickr (*FK*) [42] 是图像关系图，每个节点都将图像的描述和属性作为特征。Yelp (*YP*) 是一种社交关系图，其中客户评论者是节点，他们之间的关系则作为边。对于 YP，我们在预处理时遵循了 [90] 。GitHub (*GH*) [64] 是一种代码托管关系图。我们使用了这些数据集的原始特征。

表 II：基准数据集信息

数据集	#顶点	#边	#输入特征	拓扑	特征	稀疏性（含28层）	准确率（含28层）
Cora（CR）	0.003 M	0.01 M	1,433	0.09 MB	0.01 GB	66.1 %	76 %
CiteSeer (CS)	0.003 M	0.009 M	3,703	0.08 MB	0.05 GB	69.7%	66 %
PubMed（PM）	0.02 M	0.09 M	500	0.75 MB	0.04 GB	70.7%	77 %
NELL（NL）	0.07 M	0.25 M	61,278	2.17 MB	15.0 GB	51.0 %	64 %
Reddit（RD）	232 M	115 M	602	875 MB	0.52 GB	58.4 %	95 %
Flickr (FK)	0.09 M	0.90 M	500	7.21 MB	0.17 GB	46.5%	48 %
Yelp (YP)	716 M	14.0 M	300	109MB 0.80 GB		64.0 %	54 %
DBLP（DB）	0.02 M	0.11 M	1,639	0.87 MB 0.11GB		59.5 %	86 %
GitHub（GH）	0.04 M	0.58 M	128	4.55MB 0.02 GB		44.6 %	86 %

我们使用第 III-B 节、第 V-D 节和第V-E 节中介绍的加速器架构进行了评估。系统配置汇总于表III中。按照 [47],的说明，除非另有说明，我们使用 512KB 全局缓存。加速器的运行频率为 1 GHz。加速器配备了 32×32 基于脉动阵列的组合引擎，以及第 V-D 节中的稀疏聚合引擎；特征和权重均采用 32 位定点数。

我们使用 Verilog HDL 设计加速器，并使用 Synopsys Design Compiler 和 45 纳米 OpenPDK 库对其进行综合，然后将其缩放至 32 纳米，以匹配片上存储器模型的工艺节点。对于片上缓存，由于缺少存储单元库，我们使用 CACTI 6.5 [73]估算面积和功耗。GCNAX 的基线芯片面积为 3.95 mm^2，，采用 SGCN 的加速器面积为 4.05 mm^2，，面积开销为 2.5%，这是稀疏聚合和压缩引擎中增加逻辑所致。我们验证了其功能正确，并测得面积和功耗数值。作为比较，我们复现的 AWB-GCN 因逻辑复杂，功耗为 4.25 mm^2 。

为衡量性能，我们设计了一个用 C++ 编写的周期精确模拟器。该模型基于 SCALE- Sim [67] ，用于组合引擎中的脉动阵列；我们在此基础上扩展了一个自研模块，用于稀疏聚合引擎和压缩引擎。计算模块的执行周期已通过 HDL 设计在周期级别进行了验证。对于 DRAM 子系统的建模，我们使用 DRAMsim3 [50] 和一个 HBM2 模块。

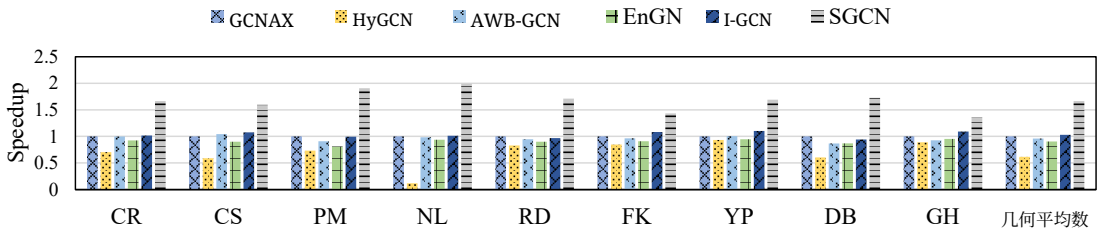


图11：GCN 加速器的性能比较。

B. 快速且节能的图卷积网络执行

性能。 图 11 表明，SGCN 的性能远超现有 GCN 加速器。在图 11中，我们将 SGCN 的性能与先前的方法进行了比较。HyGCN [80] 基于行乘积执行，并采用混合引擎。EnGN [51] 采用顶点分块和基于度感知的顶点缓存（DAVC），显式缓存高阶顶点。AWB-GCN [17] 采用基于列乘积的执行，并应用激进的负载均衡技术。此外，I-GCN [18]采用动态重排序来增强图拓扑的局部性。此外，GCNAX [47] 基于完美分块，并根据离线分析提出了优化的循环排序。在这些方法中，我们选择 GCNAX 作为基线，并报告归一化加速比，因为它在现有加速器中表现出色，且采用了与 SGCN 最相似的分块方法。

结果表明，SGCN 相较于基线 GCNAX 的几何平均加速比为 1.66× ，相较于 HyGCN 的加速比为 2.71× 。由于 GCNAX 对图拓扑（A）和中间特征（X）都采用了激进的分块策略，SGCN 的加速主要来自对特征稀疏性的利用。相较于 HyGCN 的加速主要来自两个因素：使用 BEICSR 格式减少了动态随机存取存储器访问量，以及图/特征分块带来的缓存效率提升。由于 HyGCN 不进行任何分块或切片，因此在大型图上缓存效率较低。AWB-GCN 对特征采用零值跳过，这是一种稀疏性感知方法。然而，AWB-GCN 以稠密格式存储特征，无法减少内存流量。此外，它只在组合阶段利用特征的稀疏性，而该阶段仅占 GCN 总执行时间的一小部分。通过采用稀疏格式来减少内存流量，SGCN 的几何平均性能比 AWB-GCN 高 1.73× 。与 EnGN 相比，SGCN 的加速比也达到 1.85× 。EnGN 使用的度感知顶点缓存比 HyGCN 更有效，但其有限的顶点分块仍导致缓存效率较低，因此 SGCN 相较于 EnGN 具有额外优势。

与基线相比，在 PubMed 和 NELL 数据集上分别观察到 1.91× 和 1.99× 的加速比，优于 GCNAX。PubMed 数据集的中间特征稀疏性很高，接近 70%（见图2），因此聚合阶段的加速比很高。在另一方面，NELL 数据集的特征稀疏性相对较低。然而，其输入特征宽度异常大（61,278），且极度稀疏（99.9%），因为输入特征是独热编码向量。因此，除了中间特征带来的加速比外，这一独特的输入层也带来了额外的加速比。另一方面，尽管 Cora 和 CiteSeer 的稀疏性相对较高，其加速比仍与几何平均数相近。部分原因在于数据集规模较小。当图拓扑规模较小时，组合阶段所占的相对比例会增加，从而抵消稀疏聚合带来的收益。此外，这两个图的平均度都很低：Cora 为 3.92，CiteSeer 为 2.76；而几何平均数为 10.15，最高值约为 500（Reddit）。这意味着，与其他数据集相比，对特征的随机访问次数更少，这也会抵消稀疏聚合带来的收益。

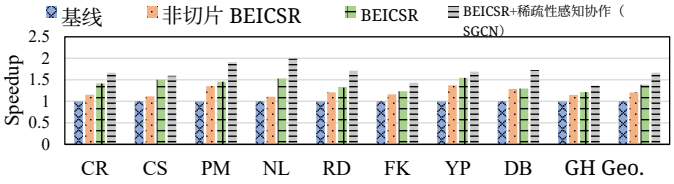


图12：消融实验。

消融实验。 图12展示了所提出的各项技术对 SGCN 性能提升的贡献。以 GCNAX 为基线，我们首先采用非切片版 BEICSR。非切片版 BEICSR 已足以利用中间特征稀疏性，但由于缺少特征矩阵切片，其数据流并非最优。因此，性能提升通常不够显著。几何平均性能提升为 20.8%。

采用切片版 BEICSR 支持特征矩阵切片后，稀疏性便可在最优数据流中得到利用。相较于基线，几何平均加速比为 38.5%，比非切片 BEICSR 高出 17.7%。

最后，我们加入稀疏性感知协作，以更好地捕捉稀疏特征不断变化的局部性。这使几何平均加速比额外提升 28.0%，整体加速比达到 1.66× 。对于聚类拓扑程度较高（DB）且邻居相似度较高（PM、RD）的图，稀疏性感知协作带来的收益更大。

能耗。 图。13 展示了能耗方面的优势。与 GCNAX、AWB-GCN 和 HyGCN 相比，SGCN 的能耗分别降低了 44.1%、44.6% 和 58.1%。能耗节省来自整个技术栈的三个部分。能耗中很大一部分来自内存访问。由于聚合阶段

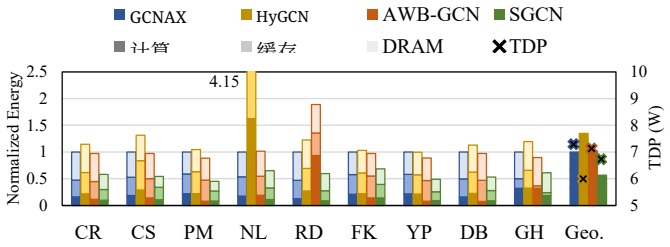


图13：SGCN能耗分解。

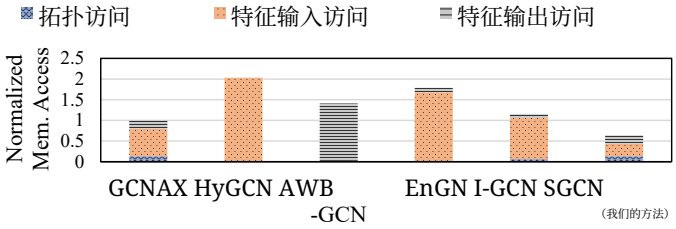


图14：Reddit（RD）的片外存储器访问分解。

由于稀疏性而需要处理的数据更少，因此片外内存和缓存的访问次数都会受到影响。此外，聚合中乘法运算次数的减少也降低了计算能耗。就峰值功耗而言，SGCN 的功耗为 6.74W，低于 AWB-GCN（7.03W）和 GCNAX（7.16W）。不过，与 HyGCN（5.94W）相比，SGCN 的峰值功耗更高；HyGCN 的架构较简单，但速度较慢。

存储器访问分解。图 14 展示了执行 RD 数据集期间的内存访问分解情况。在 HyGCN 中，大部分访问都针对特征。由于它既未采用分块，也未利用特征稀疏性，因此对特征进行了大量重复访问，而这些访问占 HyGCN 内存访问的大部分。EnGN 借助其度感知顶点缓存（DAVC）减少了部分访问。另一方面，AWB-GCN 基于列乘积数据流，每个输入特征元素只读取一次。然而，代价是需要更频繁地读写部分和，这成为内存访问的主要部分。GCNAX 通过积极分块实现了均衡的内存访问。I-GCN 也因其聚类良好的重排序方案而呈现出均衡的内存访问。相比之下，SGCN 大幅减少了内存访问量。主要的减少来自稀疏特征表示，它使特征访问量减少了 54.3%。

C. 敏感性研究

GCN 层数。在图15中，我们绘制了使用 CR、CT 和 PM 数据集时，GCN 层数对几何平均性能的影响。除了默认的 28层设置外，我们还采用了 [44], [45]中广泛使用的 7 至 112 层 GCN。在所有设置下，稀疏性基本保持不变，加速比趋势也依然存在。这表明 SGCN 带来的性能提升并非针对某一

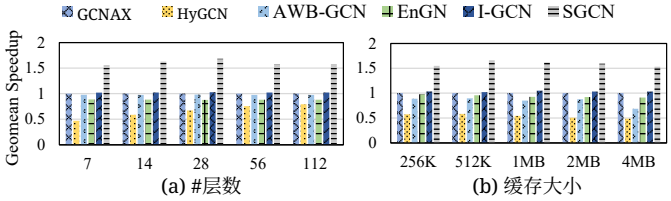


图1 15：层数和缓存大小的敏感性。

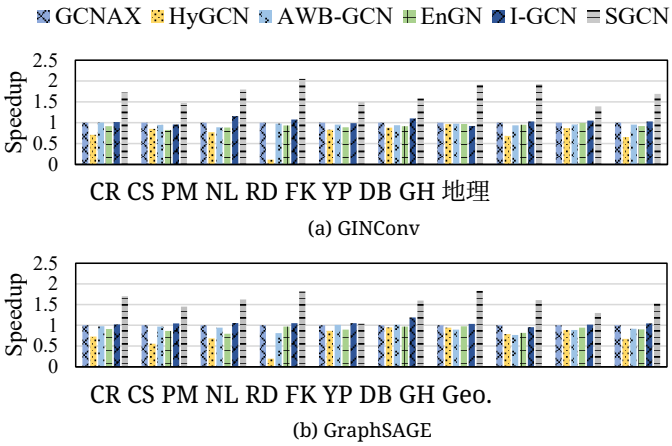


图1 16： (a) GINConv 和 (b) GraphSAGE 上的性能。

特定层数进行微调，因此可广泛应用于各种规格。

缓存大小。图115b 展示了随着缓存大小增加，加速比的变化趋势。总体而言，除非数据完全装入缓存，否则特征稀疏性带来的加速比不会受到缓存大小的显著影响。缓存较小时，稀疏性感知协作的收益会变得微乎其微，因为此时更难捕获任何局部性。然而，即使缓存更大，加速比仍相对稳定。

GCN 变体。图116 展示了另外两种 GCN 聚合变体的性能对比：GINConv [79] 和 GraphSAGE [22]。对于这两种变体，SGCN 获得的加速比与图111中的结果相近，但略有差异。与 GCN 聚合 [37],不同，GINConv 的聚合阶段不需要边权重。这˜减小了A的大小，并使聚合过程中特征矩阵所占的比例增加。由于 SGCN 能够有效减少对特征矩阵的访问，与原始 GCN 相比，其加速比略有提升。相反，GraphSAGE 对边进行随机采样，以降低计算开销。这会减少图拓扑中的有效边数，并降低聚合所占的比例。因此，与现有方法相比，SGCN 的加速比略低，但仍然显著。在 GINConv 中，SGCN 相较于 GCNAX 获得 1.69× 的加速比，相较于 HyGCN 获得 2.57×的加速比。在 GraphSAGE 中，SGCN 相较于 GCNAX 和 HyGCN 分别获得 1.53× 和 2.27× 的加速比。

切片大小的选择。采用原位压缩并支持特征矩阵切片，另一个影响是加速比会因所选

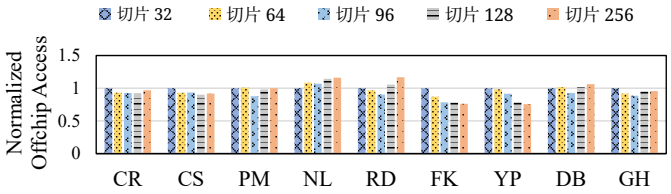


图17：SGCN切片单元大小敏感性。

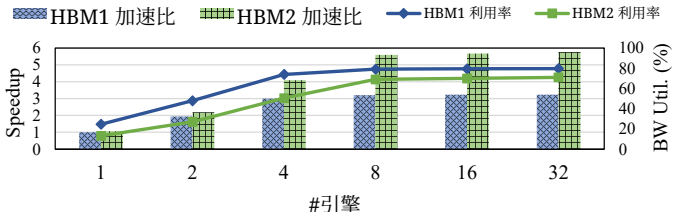


图18：SGCN内存类型与可扩展性。

单位切片大小 C而异。当切片宽度过大时，过多切片可能占用额外的缓存行。由于这会从该区域一并读取大量无效数据，因此会降低内存效率。另一方面，如果 C 过小，切片数量就会增加，从而影响输出特征访问量。尽管额外的缓存行访问次数会较少，但这可能会增加执行时间。尽管如此，图17 表明，在 $C = 32$ 到 $C = 256$的范围内，性能对切片大小并不十分敏感。根据实验，整体性能在 $C = 96$时最佳，但即使选择不佳，与基线相比仍能获得显著的加速比。

可扩展性。图。18 展示了在使用不同内存模块（HBM1 和 HBM2）的情况下，SGCN 如何随引擎数量的变化而扩展。我们将计算分配给多个引擎。在提供足够带宽时，增加引擎数量可带来近乎线性的加速比，最多可扩展至约八个引擎，体现出良好的可扩展性。可扩展性在约 16 个引擎时开始趋于饱和，此时系统已接近内存模块的最大带宽。

VII. 讨论

A. 稀疏度范围

根据第II节中的观察结果，我们设计了 BEICSR，使其适用于稀疏性约为 50% 的情况。事实上，我们考察的所有真实世界数据集都呈现出 40% 到 80% 的中等稀疏性，在这一范围内，SGCN 的优势十分显著。尽管如此，仍需明确 BEICSR 在哪些稀疏性范围内优于现有技术。虽然这种做法高度人工化，但我们在图19中绘制了 SGCN 在不同稀疏性水平下的几何平均加速比。我们根据目标稀疏性，为每一层随机生成合成输入激活值。如图所示，SGCN 在几乎所有稀疏性范围内都表现更佳。只有在稀疏性低于 5% 时，稠密格式才更优，因为 BEICSR 需要额外的位图索引。压缩稀疏行的盈亏平衡点

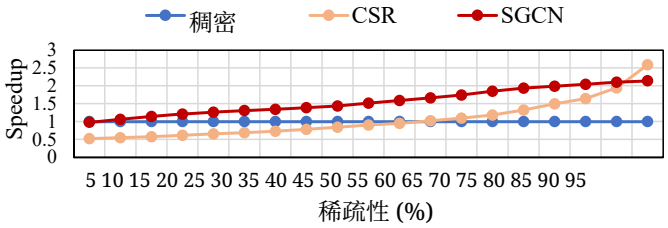


图19：不同特征稀疏性下的性能比较。

出现在稀疏性超过 90% 时，此时压缩稀疏行中的列索引所占空间小于 BEICSR 的位图索引所占空间。

B. 对未来*GCN*的适用性

如图19所示，SGCN 在广泛的稀疏性范围内都有优势。我们认为，未来的发展趋势会使稀疏性水平保持在类似范围内。激活值的稀疏性源于 ReLU函数的广泛使用。对于归一化值，ReLU后的分布将接近零均值，从而产生 ∼50 % 的稀疏性。因此，即使图卷积网络发展出不同的特性，其稀疏性水平也不太可能朝极端方向变化。

一个例外是输入层，其特征通常由人工设计。如果第一层的输入特征采用独热向量格式，稀疏性可能会非常高，例如 NELL [8]。在这种情况下，对输入特征使用 CSR 格式会更好，而 SGCN 使用专为处理 CSR 格式而设计的聚合引擎来执行组合。不过，这种技术带来的优势通常会分摊到数十层中，因此影响并不显著。

VIII. 相关工作

A. 图卷积网络 加速器

随着图卷积网络成为重要且热门的应用领域，图卷积网络的硬件加速近来受到体系结构领域的关注。HyGCN [80] 指出了对图卷积网络加速器的需求，并讨论了若干关键特性，例如混合计算模式和层间优化。此外，GNNA [6] 基于此类混合架构提出了一种用于图卷积网络加速的分块架构。EnGN [51] 讨论了对拓扑进行分块的必要性，并提出利用专用缓存来存储高阶顶点，从而实现高效的分块排序。AWB-GCN [17] 提出采用基于列的执行，并应用了多种负载均衡技术。GCNAX [47] 是一种灵活的架构，其最优数据流和缓冲区配置是基于完美分块确定的。然而，完美分块会过度配置所需的缓冲区容量，导致性能不佳。此外，作者假设稀疏数据服从均匀随机分布，而这往往与实际分布不符。SnF [87]通过采用动态分块策略，解决了GCNAX中完美分块和离线分析带来的问题。通过利用重复访问模式

进行特征分块，SnF提出了在线动态非完美拓扑分块。I-GCN [18] 提出了一种基于广度优先搜索的动态重排序方案，称为岛化。此外，它在聚合阶段重用重叠计算，以降低计算复杂度。ReGNN [9],扩展了类似的方法，识别聚合和边更新中的若干冗余，以进一步优化性能。此外，值得注意的是GCoD [89], ，它是一种面向图卷积网络的协同设计方法。与针对深度神经网络的剪枝方法类似 [39], ，GCoD会对邻接矩阵进行剪枝和聚类，并分别处理稠密区域和稀疏区域。然而，这类方法需要重新训练图卷积网络，因此只有在用户能够承担训练开销时才适用。

B. 图处理加速器

图处理是一个重要的应用领域，已有许多基于专用集成电路 [21],[55], [86], 、FPGA [40], [13], [5], 或两者兼有 [11]的加速器被提出。由于带宽需求较高，图处理通常成为存内处理 [1], [92], [100] 或存储内处理 [53], [34]的主要目标。Tesseract [1] 利用 HMC [58] 的内部带宽来加速图处理，之后又通过改进分区 [92]或执行对齐 [100]进一步完善。GraphSSD 是一种能够识别图结构的固态硬盘，而 GraFboost [34] 则在存储设备中集成了用于图处理的排序归约 FPGA 加速器。然而，它们主要面向传统图处理，并未针对 GCN执行进行优化，尤其没有针对逐顶点特征的稀疏性进行优化。

C. 稀疏深度神经网络

另一类相关研究聚焦于稀疏深度神经网络加速器。由于剪枝和 ReLU激活函数，许多权重或激活值会变为零，从而带来优化机会。一些先前研究设计了与现有硬件兼容的计算剪枝方案 [26], [54], [69]。[98], [76] 是支持稀疏运算的张量核心扩展。还有一些设计会消除零值权重 [23], [94],[97], [2], [14] ，或考虑零值激活 [56], [20], [33], [30]。有些方法面向稀疏张量代数，针对稀疏矩阵-向量乘法 [65], [4]和稀疏矩阵乘法 [96], [24], [60]。RingCNN [27]使用环代数来处理代数稀疏性。此外，一些方法提出了张量压缩的软硬件协同设计 [36], [71], [59], [85]。然而，这些方法主要针对传统卷积神经网络或嵌入层，其特性与图卷积网络有很大差异。据我们所知，这是首项针对 GCN加速器、提出利用特征稀疏性技术的工作。

*D. DNN*执行的内存带宽优化

众所周知，内存是影响 DNN加速器 [62], [29], [70], 性能的重要因素，而针对内存带宽的优化通常至关重要。一种常见方法是优化片上存储器的使用。许多加速器采用不同的数据流 [15], [12], [32],

一些研究者声称，数据分块更为重要 [84]。类似地，一些工作致力于管理片上存储器 [77], [48],[31]。另一种思路是利用深度神经网络的多层特性。将层间数据保留在片上存储器中，可以减少片外流量的带宽需求 [3], [28], [52], [35]。这些方法已成功应用于推理 [3], [28], 训练 [52], 和批归一化 [35]。在诸多区别中，我们强调，这些技术侧重于片上存储器管理，尤其是将其作为手动管理的暂存器。这些方法之所以有效，是因为传统深度神经网络具有高度规则的内存访问模式；从根本上说，它们与本工作不同。对于 GCN执行，图中的不规则访问使得采用现有 DNN加速器中的这些常见技术颇具挑战。

IX. 结论

我们提出了 SGCN，这是一种利用中间特征稀疏性的 GCN 加速器。随着图卷积网络不断向深层和残差层发展，中间特征稀疏性带来了巨大的潜力，而传统浅层图卷积网络无法利用这一潜力。

在这种情况下，我们首次提出利用特征数据中的稀疏性来优化 GCN 执行过程中的内存流量。我们首先发现，直接利用特征稀疏性的朴素尝试反而可能导致速度下降。基于这一观察，我们提出了 BEICSR，这是一种专为 GCN 执行的稀疏聚合阶段设计的稀疏特征表示格式。此外，我们还提出了稀疏性感知协作方法，以便在稀疏性变化、访问模式随之改变时更好地处理局部性。该格式的核心思想包括使用嵌入式位图索引、原位压缩，以及支持近期 GCN 加速器采用的特征切片技术。通过全面评估，我们证明，与此前最先进的 GCN 加速器相比，SGCN 能够实现更优的性能和能效。

致谢

本研究部分获得韩国国家研究基金会（NRF）项目（2022R1C1C1011307、2022R1C1C1008131）以及韩国政府（科学技术信息通信部，MSIT）资助的信息通信技术规划与评估院（IITP）项目（2021-0-00853、2020-0-01361）的支持。EDA工具由韩国集成电路设计教育中心（IDEC）提供支持。Mingi Yoo、Jaeyong Song、Jounghoo Lee和Youngsok Kim部分获得了由教育部（MOE，韩国）和韩国国家研究基金会（NRF）资助的BK21 FOUR（培育卓越研究型大学）项目的支持。

参考文献

[1] J. Ahn、S. Hong、S. Yoo、O. Mutlu和K. Choi，《用于并行图处理的可扩展存内处理加速器》，发表于ISCA，2015年。[2] J. Albericio、P. Judd、T. Hetherington、T. Aamodt、N. E. Jerger和A. Moshovos，《Cnvlutin：无无效神经元的深度神经网络计算》，发表于ISCA，2016年。[3] M. Alwani、H. Chen、M. Ferdman和P. Milder，《融合层CNN加速器》，发表于MICRO，2016年。[4] B. Asgari、R. Hadidi、T. Krishna、H. Kim和S. Yalamanchili，《ALRESCHA：轻量级可重构稀疏计算加速器》，发表于HPCA，2020年。[5] M. Asiatici和P. Ienne，《借助缓存处理数千个并发缺失的FPGA大规模图处理》，发表于ISCA，2021年。[6] A. Auten、M. Tomei和R. Kumar，《图神经网络的硬件加速》，发表于DAC，2020年。[7] P. Boldi和S. Vigna，《WebGraph框架I：压缩技术》，发表于WWW，2004年。[8] A. Carlson、J. Betteridge、B. Kisiel、B. Settles、E. R. Hruschka和T. M. Mitchell，《迈向永无止境的语言学习架构》，发表于AAAI，2010年。[9] C. Chen、K. Li、Y. Li和X. Zou，《ReGNN：冗余消除图神经网络加速器》，发表于HPCA，2022年。[10] M. Chen、Z. Wei、Z. Huang、B. Ding和Y. Li，《简单深层图卷积网络》，发表于ICML，2020年。[11] X. Chen、T. Huang、S. Xu、T. Bourgeat、C. Chung和Arvind，《FlexMiner：面向图模式挖掘的模式感知加速器》，发表于ISCA，2021年。[12] Y.-H. Chen、T. Krishna、J. S. Emer和V. Sze，《Eyeriss：用于深度卷积神经网络的节能可重构加速器》，发表于JSSC，2016年。[13] V. Dadu、S. Liu和T. Nowatzki，《PolyGraph：展现灵活性对图处理加速器的价值》，发表于ISCA，2021年。[14] C. Ding、S. Liao、Y. Wang、Z. Li、N. Liu、Y. Zhuo、C. Wang、X. Qian、Y. Bai、G. Yuan、X. Ma、Y. Zhang、J. Tang、Q. Qiu、X. Lin和B. Yuan，《CirCNN：利用块循环权重矩阵加速并压缩深度神经网络》，发表于MICRO，2017年。[15] Z. Du、R. Fasthuber、T. Chen、P. Ienne、L. Li、T. Luo、X. Feng、Y. Chen和O. Temam，《ShiDianNao：将视觉处理移近传感器》，发表于ISCA，2015年。[16] A. Fout、J. Byrd、B. Shariat和A. Ben-Hur，《利用图卷积网络进行蛋白质界面预测》，发表于NeurIPS，2017年。[17] T. Geng、A. Li、R. Shi、C. Wu、T. Wang、Y. Li、P. Haghi、A. Tumeo、S. Che、S. Reinhardt和M. C. Herbordt，《AWB-GCN：具有运行时工作负载再平衡功能的图卷积网络加速器》，发表于MICRO，2020年。[18] T. Geng、C. Wu、Y. Zhang、C. Tan、C. Xie、H. You、M. C. Herbordt、Y. Lin和A. Li，《I-GCN：通过岛化增强运行时局部性的图卷积网络加速器》，发表于MICRO，2021年。[19] M. Girvan和M. E. Newman，《社交网络和生物网络中的社区结构》，《美国国家科学院院刊》，2002年。[20] A. Gondimalla、N. Chesnut、M. Thottethodi和T. N. Vijaykumar，《SparTen：用于卷积神经网络的稀疏张量加速器》，发表于MICRO，2019年。[21] T. J. Ham、L. Wu、N. Sundaram、N. Satish和M. Martonosi，《Graphicionado：高性能、节能的图分析加速器》，发表于MICRO，2016年。[22] W. L. Hamilton、R. Ying和J. Leskovec，《大型图上的归纳式表示学习》，发表于NeurIPS，2017年。[23] S. Han、X. Liu、H. Mao、J. Pu、A. Pedram、M. A. Horowitz和W. J. Dally，《EIE：基于压缩深度神经网络的高效推理引擎》，发表于ISCA，2016年。[24] K. Hegde、H. Asghari-Moghaddam、M. Pellauer、N. Crago、A. Jaleel、E. Solomonik、J. Emer和C. W. Fletcher，《ExTensor：稀疏张量代数加速器》，发表于MICRO，2019年。[25] I. Higgins、D. Amos、D. Pfau、S. Racaniere、L. Matthey、D. Rezende和A. Lerchner，《迈向解缠结表示的定义》，arXiv预印本，2018年。[26] W. Hua、Y. Zhou、C. D. Sa、Z. Zhang和G. E. Suh，《利用动态细粒度通道门控提升CNN加速器性能》，发表于MICRO，2019年。[27] C.-T. Huang，《RingCNN：利用代数稀疏环张量实现能效型CNN计算成像》，发表于ISCA，2021年。[28] C.-T. Huang、Y.-C. Ding、H.-C. Wang、C.-W. Weng、K.-P. Lin、L.- W. Wang 和 L.-D. Chen，《eCNN：面向边缘推理的块式高度并行CNN加速器》，发表于 MICRO，2019年。[29] B. Hyun、Y. Kwon、Y. Choi、J. Kim 和 M. Rhu，《NeuMMU：支持NPU高效地址转换的架构》，发表于体系结构与编程语言操作系统支持国际会议，2020年。[30] J.-W. Jang、S. Lee、D. Kim、H. Park、A. S. Ardestani、Y. Choi、C. Kim、Y. Kim、H. Yu、H. Abdel-Aziz、J.-S. Park、H. Lee、D. Lee、M. W. Kim、H. Jung、H. Nam、D. Lim、S. Lee、J.-H. Song、S. Kwon、J. Hassoun、S. Lim 和 C. Choi，《面向三星旗舰移动SoC的稀疏感知可重构NPU架构》，发表于 ISCA，2021年。[31] T. Jin 和 S. Hong，《Split-CNN：拆分卷积神经网络中的基于窗口的运算以优化存储系统》，发表于体系结构与编程语言操作系统支持国际会议，2019年。[32] N. P. Jouppi、C. Young、N. Patil、D. Patterson、G. Agrawal、R. Bajwa、S. Bates、S. Bhatia、N. Boden、A. Borchers、R. Boyle、P.-l. Cantin、C. Chao、C. Clark、J. Coriell、M. Daley、M. Dau、J. Dean、B. Gelb、T. V. Ghaemmaghami、R. Gottipati、W. Gulland、R. Hagmann、C. R. Ho、D. Hogberg、J. Hu、R. Hundt、D. Hurt、J. Ibarz、A. Jaffey、A. Jaworski、A. Kaplan、H. Khaitan、D. Killebrew、A. Koch、N. Kumar、S. Lacy、J. Laudon、J. Law、D. Le、C. Leary、Z. Liu、K. Lucke、A. Lundin、G. MacKean、A. Maggiore、M. Mahony、K. Miller、R. Na- garajan、R. Narayanaswami、R. Ni、K. Nix、T. Norrie、M. Omernick、N. Penukonda、A. Phelps、J. Ross、M. Ross、A. Salek、E. Samadiani、C. Severn、G. Sizikov、M. Snelham、J. Souter、D. Steinberg、A. Swing、M. Tan、G. Thorson、B. Tian、H. Toma、E. Tuttle、V. Vasudevan、R. Walter、W. Wang、E. Wilcox 和 D. H. Yoon，《张量处理单元在数据中心中的性能分析》，发表于 ISCA，2017年。[33] P. Judd、A. Delmas、S. Sharify 和 A. Moshovos，《Cnvlutin^2：无需无效激活和权重的深度神经网络计算》，arXiv预印本，2017年。[34] S.-W. Jun、A. Wright、S. Zhang、S. Xu 和 Arvind，《GraFBoost：利用加速闪存进行外部图分析》，发表于 ISCA，2018年。[35] D. Jung、W. Jung、B. Kim、S. Lee、W. Rhee 和 J. H. Ahn，《重构批归一化以加速CNN训练》，发表于 SysML，2019年。[36] K. Kanellopoulos、N. Vijaykumar、C. Giannoula、R. Azizi、S. Koppula、N. M. Ghiasi、T. Shahroodi、J. G. Luna 和 O. Mutlu，《Smash：协同设计软件压缩与硬件加速索引，以高效执行稀疏矩阵运算》，发表于 MICRO，2019年。[37] T. N. Kipf 和 M. Welling，《利用图卷积网络进行半监督分类》，arXiv预印本，2016年。[38] H. Kwon、A. Samajdar 和 T. Krishna，《Maeri：通过可重构互连实现DNN加速器上的灵活数据流映射》，发表于体系结构与编程语言操作系统支持国际会议，2018年。[39] Y. LeCun、J. Denker 和 S. Solla，《最优脑损伤》，NIPS，1989年。[40] J. Lee、H. Kim、S. Yoo、K. Choi、H. P. Hofstee、G.-J. Nam、M. R. Nutter 和 D. Jamsek，《ExtraV：利用一致性加速器提升存储近端图处理性能》，pVLDB，2017年。[41] J. Leskovec、J. Kleinberg 和 C. Faloutsos，《随时间变化的图：致密化定律、收缩直径及可能的解释》，发表于 KDD，2005年。[42] J. Leskovec 和 A. Krevl，《SNAP数据集：斯坦福大型网络数据集集合》，http://snap.stanford.edu/data，2014年。[43] G. Li、M. M¨uller、B. Ghanem 和 V. Koltun，《使用1000层训练图神经网络》，发表于 ICML，2021年。[44] G. Li、M. Muller、A. Thabet 和 B. Ghanem，《DeepGCNs：图卷积网络能像卷积神经网络一样加深吗？》，发表于 ICCV，2019年。[45] G. Li、C. Xiong、A. Thabet 和 B. Ghanem，《Deepergcn：训练更深层图卷积网络所需的一切》，arXiv预印本，2020年。[46] H. Li、M. Yan、X. Yang、L. Deng、W. Li、X. Ye、D. Fan 和 Y. Xie，《通过双向融合实现图卷积网络的硬件加速》，IEEE CAL，2021年。[47] J. Li、A. Louri、A. Karanth 和 R. Bunescu，《GCNAX：一种灵活且高能效的图卷积神经网络加速器》，发表于HPCA，2021年。[48] J. Li、G. Yan、W. Lu、S. Jiang、S. Gong、J. Wu 和 X. Li，《SmartShuttle：优化深度学习加速器的片外存储访问》，发表于DATE，2018年。[49] Q. Li、Z. Han 和 X.-M. Wu，《深入理解用于半监督学习的图卷积网络》，发表于 AAAI，2018年。

[50] S. Li、Z. Yang、D. Reddy、A. Srivastava 和 B. Jacob，《DRAMsim3：周期精确且支持热仿真的DRAM模拟器》，IEEE CAL，2020年。[51] S. Liang、Y. Wang、C. Liu、L. He、L. Huawei、D. Xu 和 X. Li，《EnGN：面向大型图神经网络的高吞吐量、高能效加速器》，TC，2020年。[52] S. Lym、A. Behroozi、W. Wen、G. Li、Y. Kwon 和 M. Erez，《小批量序列化：利用层间数据重用进行CNN训练》，发表于 SysML，2019年。[53] K. K. Matam、G. Koo、H. Zha、H.-W. Tseng 和 M. Annavaram，《GraphSSD：感知图语义的固态硬盘》，发表于 ISCA，2019年。[54] W. Niu、X. Ma、S. Lin、S. Wang、X. Qian、X. Lin、Y. Wang 和 B. Ren，《PatDNN：通过基于模式的权重剪枝，在移动设备上实现实时DNN执行》，发表于体系结构与编程语言操作系统支持国际会议，2020年。[55] M. M. Ozdal、S. Yesil、T. Kim、A. Ayupov、J. Greth、S. Burns 和 O. Ozturk，《面向图分析加速器的高能效架构》，发表于 ISCA，2016年。[56] A. Parashar、M. Rhu、A. Mukkara、A. Puglielli、R. Venkatesan、B. Khailany、J. Emer、S. W. Keckler 和 W. J. Dally，《SCNN：一种用于压缩稀疏卷积神经网络的加速器》，发表于 ISCA，2017年。[57] A. Parashar、M. Rhu、A. Mukkara、A. Puglielli、R. Venkatesan、B. Khailany、J. Emer、S. W. Keckler 和 W. J. Dally，《Scnn：一种用于压缩稀疏卷积神经网络的加速器》，ISCA，2017年。[58] J. T. Pawlowski，《混合内存立方体（HMC）》，发表于 Hot Chips，2011年。[59] L. Pentecost 、 M. Donato 、 B. Reagen 、 U. Gupta 、 S. Ma 、 G.-Y. Wei 和 D. Brooks，《MaxNVM：通过稀疏编码和误差缓解，最大化DNN存储密度与推理效率》，发表于 MICRO，2019年。[60] E. Qin、A. Samajdar、H. Kwon、V. Nadella、S. Srinivasan、D. Das、B. Kaul 和 T. Krishna，《SIGMA：一种具有灵活互连、用于DNN训练的稀疏非规则通用矩阵乘法加速器》，发表于 HPCA，2020年。[61] M. K. Qureshi、A. Jaleel、Y. N. Patt、S. C. Steely 和 J. Emer，《高性能缓存的自适应插入策略》，发表于 ISCA，2007年。[62] M. Rhu 、 N. Gimelshein 、 J. Clemons 、 A. Zulfiqar 和 S. W. Keckler，《vDNN：用于可扩展、节省内存的神经网络设计的虚拟化深度神经网络》，发表于 MICRO，2016年。[63] M. Rhu、M. O'Connor、N. Chatterjee、J. Pool、Y. Kwon 和 S. W. Keckler，《压缩直接内存访问引擎：利用激活稀疏性训练深度神经网络》，发表于 HPCA，2018年。[64] B. Rozemberczki、C. Allen 和 R. Sarkar，《多尺度属性节点嵌入》，《复杂网络杂志》，2021年。[65] F. Sadi 、 J. Sweeney 、 T. M. Low 、 J. C. Hoe 、 L. Pileggi 和 F. Franchetti，《利用可扩展的多路归并并行化，高效执行大型高稀疏矩阵的稀疏矩阵向量乘法运算》，发表于 MICRO，2019年。[66] S. Salman 和 X. Liu，《稀疏性作为残差块的隐式门控机制》，发表于国际神经网络联合会议，2019年。[67] A. Samajdar、Y. Zhu、P. Whatmough、M. Mattina 和 T. Krishna，《SCALE-Sim：脉动式CNN加速器模拟器》，arXiv预印本，2018年。[68] P. Sen、G. Namata、M. Bilgic、L. Getoor、B. Galligher 和 T. Eliassi- Rad，《网络数据的集体分类》，《人工智能杂志》，2008年。[69] F. Silfa、G. Dot、J.-M. Arnau 和 A. Gonaz`alez，《循环神经网络中的神经元级模糊记忆化》，发表于 MICRO，2019年。[70] K. Siu、D. M. Stuart、M. Mahmoud 和 A. Moshovos，《卷积神经网络硬件加速器的内存需求》，发表于国际工作负载特征分析研讨会，2018年。[71] N. Srivastava、H. Jin、S. Smith、H. Rong、D. Albonesi 和 Z. Zhang，《Tensaurus：一种用于混合稀疏稠密张量计算的通用加速器》，发表于 HPCA，2020年。[72] J. Tang、J. Zhang、L. Yao、J. Li、L. Zhang 和 Z. Su，《Arnetminer：学术社交网络的提取与挖掘》，发表于 KDD，2008年。[73] S. Thoziyoor、N. Muralimanohar、J. H. Ahn 和 N. P. Jouppi，《CACTI 5.1》，2008-20，惠普实验室，技术报告，2008年。[74] K. Ueno、T. Suzumura、N. Maruyama、K. Fujisawa 和 S. Matsuoka，《超级计算机上的极大规模广度优先搜索》，发表于大数据，2016年。[76] Y. Wang、C. Zhang、Z. Xie、C. Guo、Y. Liu 和 J. Leng，《双侧稀疏张量核心》，发表于 ISCA，2021年。

[75] S. Vashishth，《用于自然语言处理的神经图嵌入方法》，arXiv 预印本，2019年。[77] X. Wei、Y. Liang 和 J. Cong，《通过层感知内存管理克服基于FPGA的DNN加速器中的数据传输瓶颈》，发表于 DAC，2019年。[78] C.-J. Wu、A. Jaleel、W. Hasenplaugh、M. Martonosi、S. C. Steely Jr 和 J. Emer，《SHiP：用于高性能缓存的基于签名的命中预测器》，发表于 MICRO，2011年。[79] K. Xu、W. Hu、J. Leskovec 和 S. Jegelka，《图神经网络的能力有多强？》，arXiv 预印本，2018年。[80] M. Yan、L. Deng、X. Hu、L. Liang、Y. Feng、X. Ye、Z. Zhang、D. Fan 和 Y. Xie，《HyGCN：一种采用混合架构的GCN加速器》，发表于 HPCA，2020年。[81] P. Yanardag 和 S. Vishwanathan，《深度图核》，发表于 KDD，2015年。[82] C. Yang、R. Wang、S. Yao、S. Liu 和 T. Abdelzaher，《重新审视深层GCN中的过平滑》，arXiv 预印本，2020年。[83] J. Yang、J. Lu、S. Lee、D. Batra 和 D. Parikh，《用于场景图生成的图R-CNN》，发表于 ECCV，2018年。[84] X. Yang、M. Gao、J. Pu、A. Nayak、Q. Liu、S. E. Bell、J. O. Setter、K. Cao、H. Ha、C. Kozyrakis 和 M. Horowitz，《DNN数据流选择被高估了》，arXiv 预印本，2018年。[85] Y. Yang、J. Emer 和 D. S´anchez，《SpZip：为不规则应用实现高效数据压缩的架构支持》，发表于 ISCA，2021年。[86] P. Yao、L. Zheng、Z. Zeng、Y. Huang、C. Gui、X. Liao、H. Jin 和 J. Xue，《面向图挖掘应用的局部性感知节能加速器》，发表于 MICRO，2020年。[87] M. Yoo、J. Song、H. Lee、J. Lee、N. Kim、Y. Kim 和 J. Lee，《切片与锻造：更充分地利用图卷积网络加速器的缓存》，发表于 PACT，2022年。[88] M. Yoo、J. Song、J. Lee、N. Kim、Y. Kim 和 J. Lee，《通过特征切片和自动瓦片变形，更充分地利用GCN加速器的缓存》，发表于 IEEE CAL，2021年。[89] H. You、T. Geng、Y. Zhang、A. Li 和 Y. Lin，《Gcod：通过专用算法与加速器协同设计实现图卷积网络加速》，arXiv 预印本，2021年。[90]H. Zeng、H. Zhou、A. Srivastava、R. Kannan 和 V. Prasanna，《GraphSAINT：基于图采样的归纳学习方法》，发表于 ICLR，2020年。[91] J. Zhang 和 L. Gruenwald，《规范不规则性：面向GPU上图数据的基于位图且可移植的稀疏矩阵乘法》，发表于 GRADES-NDA，2018年。[92] M. Zhang、Y. Zhuo、C. Wang、M. Gao、Y. Wu、K. Chen、C. Kozyrakis 和 X. Qian，《GraphP：通过高效数据划分减少基于PIM的图处理中的通信开销》，发表于 HPCA，2018年。[93] M. Zhang 和 Y. Chen，《基于图神经网络的链接预测》，发表于 NeurIPS，2018年。[94] S. Zhang、Z. Du、L. Zhang、H. Lan、S. Liu、L. Li、Q. Guo、T. Chen 和 Y. Chen，《Cambricon-X：一种稀疏神经网络加速器》，发表于 MICRO，2016年。[95] X. Zhang、C. Li、H. Wang 和 D. Wang，《一种采用自适应插入和再引用预测的缓存替换策略》，发表于 ISCA，2010年。[96] Z. Zhang、H. Wang、S. Han 和 W. J. Dally，《SpArch：高效的稀疏矩阵乘法架构》，发表于 HPCA，2020年。[97] X. Zhou、Z. Du、Q. Guo、S. Liu、C. Liu、C. Wang、X. Zhou、L. Li、T. Chen 和 Y. Chen，《Cambricon-S：通过软硬件协同方法解决稀疏神经网络中的不规则性》，发表于 MICRO，2018年。[98] M. Zhu、T. Zhang、Z. Gu 和 Y. Xie，《稀疏张量核心：面向现代GPU上向量级稀疏神经网络的算法与硬件协同设计》，发表于 MICRO，2019年。[99] X. Zhu、W. Han 和 W. Chen，《GridGraph：使用两级分层划分在单机上进行大规模图处理》，发表于 USENIX ATC，2015年。[100] Y. Zhuo、C. Wang、M. Zhang、R. Wang、D. Niu、Y. Wang 和 X. Qian，《GraphQ：可扩展的基于PIM的图处理》，发表于 MICRO，2019年。