切片与锻造：更好地利用图卷积网络加速器的缓存

Mingi Yoo*延世大学，韩国首尔
skys7297@yonsei.ac.kr

Jaeyong Song*延世大学，韩国首尔
jaeyong.song@yonsei.ac.kr

Hyeyoon Lee 延世大学，韩国首尔
hylee817@yonsei.ac.kr

Jounghoo Lee 延世大学，韩国首尔
jounghoolee@yonsei.ac.kr

Namhyung Kim 三星电子，韩国华城
namhyungk11@gmail.com

Youngsok Kim 延世大学 首尔，韩国
youngsok@yonsei.ac.kr

Jinho Lee†首尔大学 首尔，韩国
leejinho@snu.ac.kr

arXiv:2301.09813v1 [cs.LG] 24 Jan 2023

摘要

图卷积网络（GCN）日益受到欢迎，因为它们能够处理多种此前的深度神经网络难以支持的数据格式。为图卷积网络设计硬件加速器的一项关键挑战，在于其数据访问模式规模庞大且具有随机性，这大大降低了有限片上缓存的有效性。为缓解不规则数据访问、提升缓存效率，先前研究通常采用传统图处理应用中的顶点分块技术。尽管这些方法能够有效提升缓存效率，但它们往往对分块配置十分敏感，而最优配置很大程度上取决于目标输入数据集。此外，现有方案需要通过反复试错进行手动调优，或依赖次优的解析模型。

本文提出切片与锻造（*SnF*），这是一种高效的图卷积网络硬件加速器，可大幅提升有限片上缓存的有效性。SnF 选择了一种名为特征切片的分块策略，将特征拆分为垂直切片，并在执行的最外层循环中处理这些切片。这一选择使得针对不规则图数据的相同计算模式在多轮处理中重复出现。利用这种重复性，SnF 能够动态调整分块大小。实验结果表明，与先前工作相比，SnF 在多引擎设置下的几何平均性能可提升 1.73× 倍，在小规模设置下的几何平均性能可提升 1.46× 倍，且无需进行离线分析。

CCS 概念

• **计算机系统组织** → **神经网络**；**单指令、多数据**。

关键词

图卷积网络、加速器、缓存

本文为作者预印本版本，论文将发表于 PACT'22 会议论文集。

* 共同贡献 † 通讯作者。

1 引言

图卷积网络（GCN）被视为新一代深度神经网络（DNN），因为它们可应用于各种呈现不规则模式的数据。尽管其适用范围广泛，从传统图问题 [16, 81] 到现代深度神经网络任务 [70], ，传统深度神经网络加速器却无法高效加速图卷积网络。与传统深度神经网络不同，图具有高度稀疏的结构，因此需要从根本上不同的加速方案。尽管一些利用深度神经网络稀疏性的硬件加速器 [35, 54, 83] 或许能够处理图的高度稀疏结构，但这类加速器仍不适用于图卷积网络，因为图中的非零值比例（<0.01 %）远低于稀疏深度神经网络（10%–90%）。

为了高效加速图卷积网络，先前的研究提出了新型硬件加速器，以利用图卷积网络的各种独特特性。特别是，由于图卷积网络的工作集大小较大，且内存访问模式高度不规则，硬件加速器应高效利用有限的片上缓存。将图卷积网络视为一系列级联的稀疏-稠密矩阵乘法时，一种常用技术是循环分块与重排序，如 [17, 46, 50] 所示。循环分块将给定的图卷积网络划分为多个瓦片，与稀疏矩阵-向量乘法方法 [26, 89] 中的做法类似。通过将工作集限制在特定区域内，循环分块能够提高缓存效率，但代价是增加重复次数。

然而，循环分块的难点在于分块大小并非易于配置。对于具有大量随机访问的图卷积网络，最优设置会因数据集而异，无法静态确定。因此，目前获取最优分块配置的方法是通过反复试错进行手动调优。虽然有一种方法可以通过解析搜索来限制性地应用分块配置 [46]，但它忽略了图的稀疏性，往往会得到较差的结果。

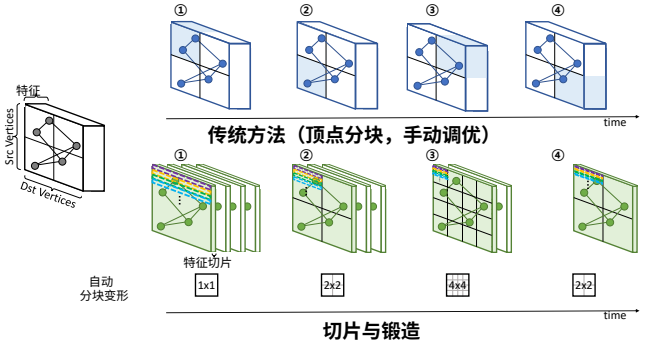


图 1：切片与锻造方案的概念示意图。

在本工作中，我们提出了切片与锻造（*SnF*），这是一种支持图卷积网络配置在线调优的技术。图 1 从概念上展示了所提出的方案。SnF 基于特征切片运行，该方法将特征矩阵拆分为多个垂直条带，并将其循环置于最外层。这会在随机图上产生重复的访问模式，从而能够逐步调优配置，使其经过几轮后达到接近最优的配置。

利用这些重复模式，我们提出了一种通过自动瓦片变形高效调优顶点分块配置的方法。在顶点分块中，邻接矩阵被划分为多个瓦片，而性能对瓦片大小十分敏感。借助自动瓦片变形，我们采用类似凸优化的方法来寻找最优瓦片大小。

尽管特征切片本身是一种循环分块技术，长期以来已在稀疏矩阵乘法 [2, 86]或图卷积网络 [46], 的背景下得到研究，但据我们所知，尚无研究利用特征切片产生的重复模式进行动态配置。除了实现基于重复模式的动态配置外，我们还表明，特征切片本身通常也是一种有利的循环分块选择。

我们的贡献可总结如下：

- 我们发现，特征切片会在随机图数据上产生重复模式，并提出切片与锻造（SnF），利用这些模式实现高效的动态调优。
- 我们提出自动瓦片变形，可在运行时动态调整顶点分块的数量。
- 我们提出一种策略，将 SnF 扩展至多芯片模块扩展。
- 我们对整合上述技术的 SnF 进行了全面评估，并探讨其优势。

2 背景

2.1 图卷积网络

考虑到多种变体， [23, 39, 74], GCN 层的推理 *l*可表述如下：

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

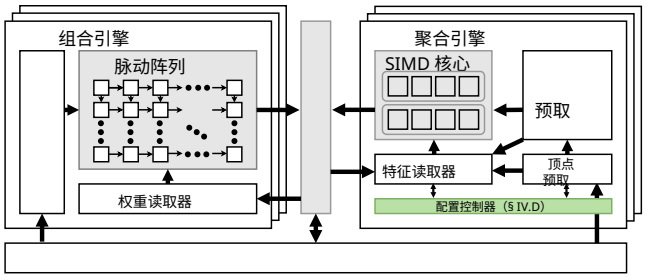


图 2：基线图卷积网络加速器架构。

在上述公式中， $\tilde{A} \in \mathbb{R}^{|V|\times|V|}$ 是归一化拉普拉斯矩阵[1]。$X^{(l)} \in \mathbb{R}^{|V|\times|F^{(l)}|}$ 是第 l 层的特征矩阵，其中每一行表示与一个顶点相关联的特征。$W^{(l)} \in \mathbb{R}^{|F^{(l)}|\times|F^{(l+1)}|}$ 是第 l 层的可训练权重矩阵，也是图卷积网络中可训练的部分。 $\sigma(\cdot)$ 是激活函数，通常使用修正线性单元（ReLU）。图卷积网络通常由三到五层组成，因为众所周知，层数过多会难以训练。

式（1）包含两次矩阵乘法。第一次是 $\tilde{A}$和 $X^{(l)}$ 或 $\tilde{A}$ 和 $X^{(l)} \cdot W^{(l)}$ 之间的矩阵乘法，称为聚合 阶段。由于 $\tilde{A}$的每一行都表示一个顶点的邻居列表，因此将该行与矩阵 $X^{(l)}$ 相乘，本质上就是对相邻顶点的特征进行“聚合”。求和是最常用的聚合函数，但也可以使用许多求和以外的聚合函数（例如，最大值）。尽管如此，其底层计算模式与矩阵乘法相同。第二次是 $\tilde{A} \cdot X^{(l)}$ 和W或 $X^{(l)}$ 和W之间的矩阵乘法，称为组合阶段。由于整行中的元素都代表属于同一个顶点的特征，因此这种乘法不会在顶点之间传递任何信息，而只是“组合”该顶点所包含的特征。换言之，它不需要顶点间通信。对于每个顶点，可以将其视为特征向量通过一个全连接层，其中每个顶点共享相同的权重矩阵。

如 [75], 所述，聚合阶段通常会成为 GCN执行的主要瓶颈。与本质上是稠密矩阵乘法的组合不同，图拓扑结构（A）的高~稀疏性以及聚合阶段较低的算术密度，使其具有极高的内存密集性。此外，由于难以优化，许多研究主要聚焦于聚合阶段。值得注意的是，如式（1）所示，聚合与组合的顺序不会影响 GCN 层的结果。然而，根据W的形状，特征向量的宽度可能增大或减小，这可用于缩短聚合 [45]的执行时间。

2.2 BaselineGCN 加速器

图 2 展示了一个基线 GCN 加速器，它由聚合引擎和组合引擎组成，与之前的工作类似 [8,

[1] $\tilde{A} = I$ – 归一化拉普拉斯矩阵是邻接矩阵的一种形式，$D^{-\frac{1}{2}}AD^{-\frac{1}{2}}$ D 定义为//其中为度矩阵。

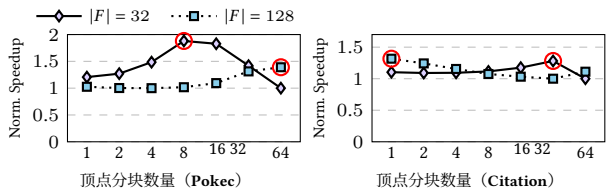


图 3：图卷积网络对顶点分块数量的敏感性。性能最佳的顶点分块数量以红色圆圈标出。

38, 50, 75], 其中，绿色组件是新增部分，将在第 4.3 节中讨论。组合引擎的核心是一个脉动阵列，用于支持高效的矩阵乘法。与以往基于脉动阵列的深度神经网络加速器类似 [34, 51], ，这里采用双缓冲方式设置了两个本地存储器，持续向核心提供输入。属性缓冲区 X和权重读取器分别提供W。组合引擎的输出由聚合引擎读取，以便在聚合函数中进行处理。

聚合阶段与经典图处理类似，但每次访问边时，都需要处理一行较宽的 X，将其与对应的边权重（即 $\tilde{A}_{i,\ j}$）相乘并累加。单指令多数据核心执行乘累加操作。为了在不发生停顿的情况下为单指令多数据核心供给数据，两个预取模块会在图拓扑中提前运行，使特征读取器始终能够即时读取特征矩阵 X。

图拓扑以压缩稀疏行（CSR）格式存储，以利用数据的稀疏性。顶点预取单元读取行指针，并将其传递给边预取单元和特征读取器。接着，边预取单元读取 CSR 的列索引，并将其发送给特征读取器和单指令多数据核心。特征读取器收到一条边后，会收集特征矩阵 X 的各行。由于边具有稀疏性，读取模式高度随机，使得聚合阶段内存密集且难以优化。

缓解聚合阶段内存瓶颈的一种方法是使用大型片上全局缓存。它由加速器中的多个引擎共享，可临时存储最近访问过的特征。然而，由于特征矩阵很宽，其宽度通常达到数百甚至数千，工作集通常远超全局缓存的容量，因此难以利用局部性。

3 动机示例

在图卷积网络执行中，最常用的减少内存访问的循环分块技术是顶点分块，该技术改编自 [17, 46, 50]。采用顶点分块时，邻接矩阵会被拆分成多个瓦片，并按顺序处理。这样一来，访问特征的范围就被限制在一个瓦片内，从而显著缩小缓存工作集（因此也能获得更多缓存命中）。然而，更高的缓存命中率是以多次访问顶点特征为代价的，无论是读取还是写入特征都如此。因此，选择合适的瓦片数量对实现高性能至关重要。

图 3 展示了顶点瓦片数量的敏感性，测试所用的图数据集为 Pokec 和 Citation（详情见表1），每个顶点的特征宽度分别为 32 和 128。Y 轴表示加速比，以所考察的瓦片数量中速度最慢者为基准进行归一化。

我们从图中得到两个观察结果：性能对瓦片数量很敏感，而且最佳顶点瓦片数量因数据集和特征宽度而异。使用默认的 1×1 个顶点瓦片时，由于缓存未命中，大量内存访问次数都花在了重复读取顶点特征上。随着瓦片数量增加，工作集大小会减小，因而更容易被缓存容纳。同时，由于重复写入，输出特征访问次数也会增加[2]。结果表明，最佳配置与最差配置之间的性能差距变化很大，在示例中最高可达 1.9×（Pokec, $|F| = 32$）。此外，输入访问量与瓦片数量之间的关系取决于邻接矩阵所呈现的随机访问模式。由于无法事先知道最佳设置，人们只能估算一个设置，而这很容易导致性能较差；或者多次执行以找到最佳配置，但这会违背实现加速比的初衷。

本文提出将特征切片作为在线分块配置的一种可能途径。利用特征切片数据流中发现的重复计算模式，可在多轮迭代中逐步找到最优配置。所提出的方案不仅搜索最优顶点分块，还搜索对图卷积网络加速器实现多引擎扩展至关重要的工作分配策略。

4 切片与锻造

4.1 特征切片数据流

许多已有的图卷积网络加速工作都基于行乘积 [8, 46, 50, 75]或列乘积 [17]。本节将介绍 SnF 的数据流选择：将特征切片置于最外层循环，以包围顶点分块。

图 4（左）展示了基于基本行乘积数据流的聚合过程（最内层循环输出一行）。彩色条带 $\tilde{A}$ 表示拓扑矩阵中的非零元素（即边），而 X中的水平条带则是要从对应颜色的边聚合的特征向量元素。在行乘执行模型中，1 最外层循环遍历 $\tilde{A}$中的顶点。2 下一层循环遍历边，并 3 聚合相应的特征。

该数据流的内存访问模式具有以下特征：首先，˜的读取是顺序进行的，且只进行一次。其次，邻近顶点的特征会从整个特征矩阵中随机读取$A\ X\ (I)$，其中每条边 $\tilde{A}$都会触发对特征矩阵一行的读取。最后，输出特征 $X\ (I+1)$ 写入内存也是顺序进行的，且只进行一次。根据以上特征，性能瓶颈在于读取特征矩阵 $X\ (I)$。如果全局缓存足够大，能够容纳这些特征，就可以通过全局缓存减少内存访问；命中率则取决于边的模式。

[2]事实上，也可以通过改变分块遍历的顺序来提高输入访问重复度。不过我们省略这种情况，因为它要求完美缓存。

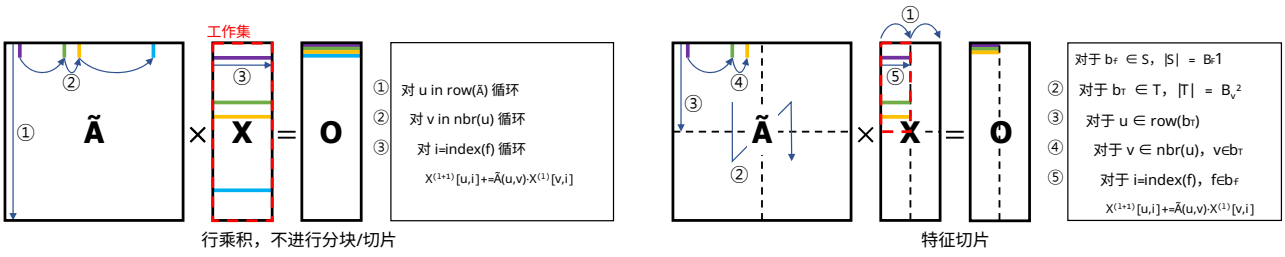


图 4：基于行乘积的 GCN 处理（左）和特征切片（右）。

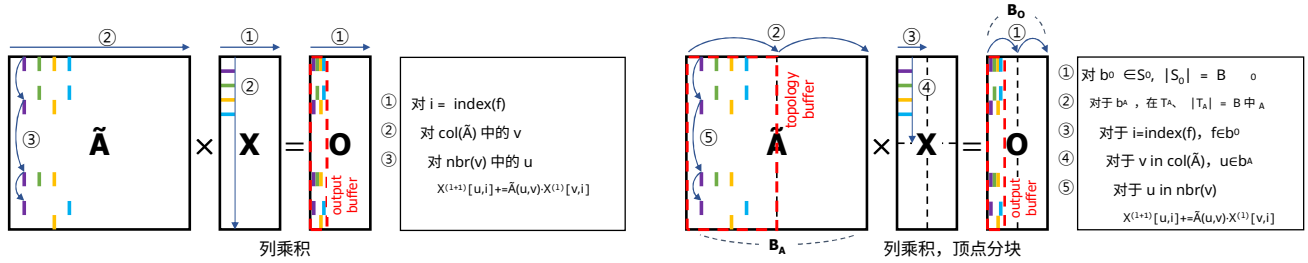


图 5：基于列乘积的执行（左）及其分块（右）。

当特征矩阵远大于全局缓存容量时，未命中率会接近 100%，而循环分块是缓解这一问题的常用方法。尽管可选配置很多，图 4（右）展示了 SnF 采用的一种方案，称为 特征切片。它将 $\tilde{A}$ 划分为 $B_V \times B_V$ 个顶点分块，并将特征矩阵 X 划分为多个由垂直条带构成的 切片。如图所示，最外层的两个循环（1 和2）遍历特征切片和顶点分块。内层循环与图 4（左）中的循环相同。从红色虚线可以看出，工作集被特征切片的数量 B_F 和瓦片数量 B_V所划分，代价是拓扑结构需要读取 A 次，特征矩阵需要读取 B_F X 次。

当 B_V和 B_F 足够大，以至于所有瓦片都能完美地放入片上内存时，只剩下冷未命中。在本文中，我们将这种策略称为完美分块。虽然完美分块可以最大限度地提高缓存效率，但 B_V所需的值通常过大。此外，它忽略了图拓扑的稀疏性，并将计算视为稠密矩阵乘法，往往导致性能不佳。

列乘积是另一种用于缓解缓存未命中的数据流方案。图 5（左）展示了列乘积执行模型。 1 最外层循环遍历特征向量（即 X 的列）中的每个元素。 2 下一层循环访问 $\tilde{A}$ 的各列。 3 最后，访问所选列 A 中的非零元素，并将相乘后的值写入输出矩阵的相应位置。

由于 X 的两个维度位于外层循环中，X 只需访问一次。然而，这需要按照特征长度访问邻接矩阵。为缓解这一问题，可以像图 5（右）所示，使用缓存来缓存邻接矩阵。当A 和 O 分别被划分为 B_A 和 B_O 个垂直条带时，缓存中存储的 A 的一条条带会被重复使用多次，从而提升

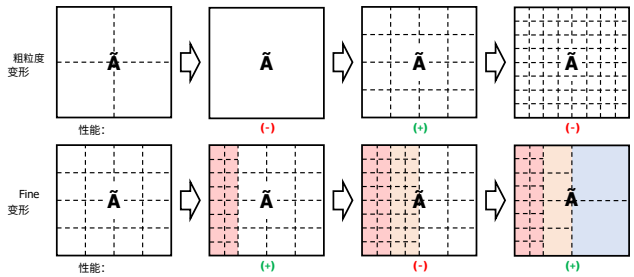


图6：瓦片自动变形流程

g.

性能。但这需要额外的缓存容量，并增加了对输出矩阵重复访问的开销。

在 SnF 中，我们将特征切片访问循环置于最外层（1）。尽管两个循环的顺序对工作集大小影响不大，仅会略微改变访问模式，但需要强调的是，这种特定选择会在随机数据访问中形成重复模式2 - 5 。这为图卷积网络带来了独特的机会：可以在循环1 的迭代过程中调整内层循环配置，因为不同特征切片之间并无差异，而顶点分块则不同。请注意，这种技术不适用于经典图分析，因为特征宽度通常过窄（例如，PageRank 的特征宽度为 1）。

B_F的大小必须经过设定，使切片宽度 $|F|/B_F$至少达到 64 B，即主存和缓存行的粒度，以保持空间局部性。SnF 将切片宽度设为 64 字节（例如， $|F| = 256$对应的 $B_F = 16$）。

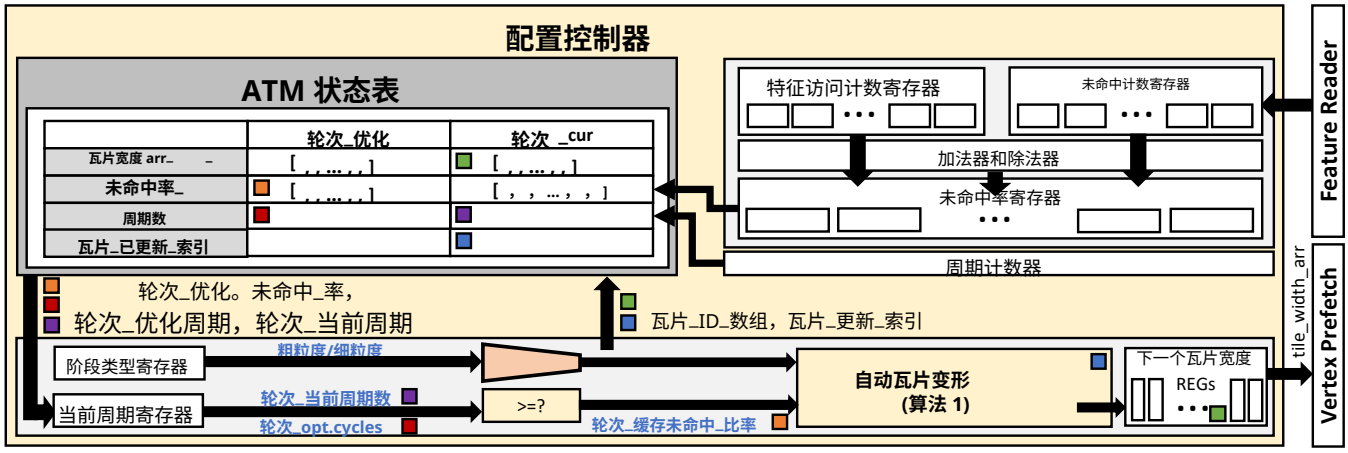


图 7：支持自动瓦片变形的配置控制器架构。

4.2 自动瓦片形态转换

如第3节所示，选择合适数量的瓦片对于实现高性能至关重要，但若不采用代价高昂的反复试验过程，就没有高效的方法来进行最优选择。我们提出了自动瓦片变形，这是一种利用重复计算模式，在运行时逐步配置顶点分块的方法。

采用特征切片数据流（第4.1节），拓扑数据 $\tilde{A}$ 会以完全相同的模式被遍历多（B_F）次（图 4（右）中的 2-5）。在执行的每一轮中——处理 X 的一个切片——自动瓦片变形会在运行时动态调整顶点分块，以达到接近最优的状态，而无需进行离线设计空间探索。虽然也可以通过将顶点分块作为外层循环，并以 $|F|$ 作为参数进行调整，来实现类似的动态方法，但每个瓦片的处理模式都不相同，因此某个瓦片的调优结果无法用于下一个瓦片。

图 6 展示了自动瓦片变形的一个示例流程，包含两个阶段：粗粒度变形和精细变形。粗粒度变形（上方）假设性能是瓦片数量的凸函数。从默认设置开始（图中为 2×2），将每个维度的瓦片数量加倍后测量性能，再将每个维度的瓦片数量减半。在性能改善的方向上，继续执行相同的加倍或合并操作，直到观察到性能下降；随后将瓦片数量回退到上一次迭代的值（示例中为 4×4），并终止粗粒度变形。

随后，精细变形（下图）从粗粒度变形阶段的最佳分区开始。精细变形的目标是使用异构尺寸瓦片，从而考虑图分布不均衡的情况。现实世界图有时通过爬取收集，爬取依据广度优先搜索顺序进行。此外，由于小世界现象，[41], 密集簇往往会出现。这些簇的访问实例数量较多，与图中较稀疏的区域相比，可能需要更小的工作集。我们将分块限制为使用可变尺寸的垂直条带和正方形瓦片，以缩小搜索空间的规模，从而有效地将精细变形转化为一维划分问题。

精细变形会记录每个瓦片的命中率，并根据这些命中率确定访问顺序。按照命中率从低到高（红色 → 橙色）的顺序，每次迭代将块的垂直条带进一步拆分为大小减半的瓦片，每次处理一列。与粗粒度变形类似，此过程会持续进行，直到性能下降，并回滚导致性能下降的更改。随后，同样按照命中率从高到低（蓝色）的顺序，将相邻两列合并，直到性能下降。虽然还可以进一步考察将一个瓦片拆分为 4×4 个瓦片，或将 4×4 个瓦片合并为一个瓦片，但我们发现，在数据集中没有任何配置能通过这种方式进一步加快执行速度。

在实现中，我们定义了单位瓦片大小，并以该单位记录每个垂直条带的长度。实际应用中，可以将单位瓦片设置为恰好适配全局缓存的瓦片大小。本文将拓扑结构划分为 64×64 个单位瓦片。最坏情况下，最多会有 64 个单位宽度条带，存储这些条带所占容量不到 1 KB。

从经验来看，自动瓦片变形能够找到接近静态最优方案的解。为避免决策过程处于关键路径上，我们根据截至块的倒数第二列的测量结果来决定下一次迭代的策略。然而，最初几个特征切片采用非最优分块执行会带来运行时开销。当 B_F 过小时，这部分开销所占比例会相对较大，这也是倾向于选择较高 B_F的另一个原因。更多讨论请参见第 6.12 节。

4.3 切片与锻造架构

为支持自动瓦片变形，SnF 在图 2 所示的加速器中新增了一个名为 配置控制器 的硬件组件。控制器的详细架构如图 7 所示。

配置控制器的总体任务是收集特征读取器提供的统计信息、做出配置决策，并将决策传递给顶点预取单元。对于每一轮迭代

B_F，控制器会累计单位瓦片垂直条带中每个特征的访问次数和未命中次数，以计算未命中率。该轮结束后，控制器会将这些未命中率和周期数更新到*ATM* 状态表中。基于当前轮（round_cur）和已存储的最优轮（round_opt）的信息，控制器调用算法 1，为下一轮生成顶点分块大小的决策。

算法 1自动瓦片变形

```
输入:
    round_cur, round_opt: 当前/最佳轮次的配置
    atm_direction: 当前 ATM 变形方向标志（HALVING/MERGING）
    phase: 当前 ATM 阶段标志（COARSE/FINE）
输出:
    new_tile_width_arr: 由 ATM 决定的下一个瓦片宽度数组
1: 如果 round_cur.cycles < round_opt.cycles 则// 如果更快，则更新为最优设置
2:  round_opt ← round_cur
3: 否则如果 phase = COARSE 则 // 如果更慢，则进入精细变形
4:     phase ← FINE
5: 否则 // 对于其余情况，这是我们找到的最优设置。
6: 返回 round_opt.tile_width_arr
7: 如果 phase = COARSE 则
8:     如果 atm_direction = HALVING 则 // 增加瓦片的数量。
9:         对于 所有瓦片 round_cur.tile_width_arr 执行
10:            追加 减半的 瓦片到 new_tile_width_arr。 // 例如 [4,4] → [2,2,2,2]
11:    否则如果 atm_direction = MERGING 则 // 减少瓦片的数量。
12:        对于 round_cur.tile_width_arr 中一半的瓦片执行
13:            追加 合并后的瓦片至 new_tile_width_arr。 // 例如 [2,2,2,2] → [4,4]
14: 否则，如果 phase = FINE，则
15:    如果 atm_direction = HALVING，则// 增加瓦片数量。
16:        找到未命中率最高的瓦片并将其二分。
17:        将二分后的瓦片和其他瓦片追加至 new_tile_width_arr。
18:    否则如果 atm_direction = MERGING 那么// 减少瓦片的数量。
19:        查找未命中率最低的瓦片，并与相邻瓦片合并。
20:        将合并后的瓦片和其他瓦片追加到 new_tile_width_arr。
21:    round_cur.tile_updated_idx ← index
22: round_cur.tile_width_arr ← new_tile_width_arr
23: 返回 new_tile_width_arr
```

算法 1 展示了自动瓦片变形的简化流程。控制器通过 $tile\_width\_arr$ 管理分块，该数组记录每个垂直条带由多少个单位瓦片组成。例如，它存储连续 32 个 2 来表示 32×32 个瓦片，并存储连续 16 个 4 来表示 16×16 个瓦片。自动瓦片变形首先更新状态表中的最优值（$round\_opt$）（第 2 行）。随后，控制器进入粗粒度变形阶段。假设在前几轮中已发现能够提升性能的方向$atm_direction$ 是将瓦片大小减半（算法中省略），那么新的 $tile\_width\_arr$ 将填入当前轮次数组中原始元素数量两倍、数值为原来一半的元素（第 8–10 行）。合并方向的工作方式类似（第 11–13 行）。在精细变形阶段，控制器根据未命中率，找出应与相邻瓦片合并或应减半的目标瓦片（t_id）（第 14–21 行）。确定$tile_width_arr$ 后，这些分块信息会传递给顶点预取单元。

配置控制器为 64 个垂直单元列中的每一列存储访问、未命中和比率寄存器。此外，状态表中的 $tile_width_arr$和 $miss\_ratio$ 是长度为 64 的数组，分别对应每个 $round\_opt$、$round\_cur$，以及顶点预取的输出端。因此，若使用 4 字节元素，寄存器和状态表的总大小仅约为 1.5 KB，与全局缓存相比微不足道。

4.4 数据流成本比较

SnF 采用的特征切片数据流隐式地优先增加切片数量 B_F，而非增加顶点分块数量 B_V。虽然这种选择确实会在一定程度上限制循环分块，但我们通过分析成本模型表明，通常优先采用前者是一种有利的策略。

使用第 2.1 节中的术语，并以 $|E|$ 表示边的数量，图 4（左）中的行乘执行所产生的内存访问成本可建模如下。

$$M_{Row} = (|V| + |E|) + m(|V||F|) \cdot |E||F| + |V||F|. \tag{2}$$

第一项和第三项分别表示顺序读取 CSR 格式的拓扑数据以及写入输出矩阵的开销。另一方面，第二项用于建模对特征矩阵的随机读取。对于拓扑结构中的每条边（$|E|$），都会读取特征矩阵的一行（$|F|$）。假设未命中率 m（x）是工作集大小 x 的函数，则第二项表示经全局缓存过滤后的访问。

应用循环分块后，通过减小工作集大小，可以缓解第二项所代表的瓶颈。使用图 4（左）中的分块方式，模型变为：

$$M_{FS} = B_F(B_V|V| + |E|) + m\left(\frac{|V||F|}{B_F B_V}\right) \cdot |E||F| + B_V|V||F|. \tag{3}$$

工作集大小除以 $|B_F|$ 和 $|B_V|$ 的乘积，其中这两个因子分别对应读取拓扑数据和写入输出矩阵时因重复而产生的次数。

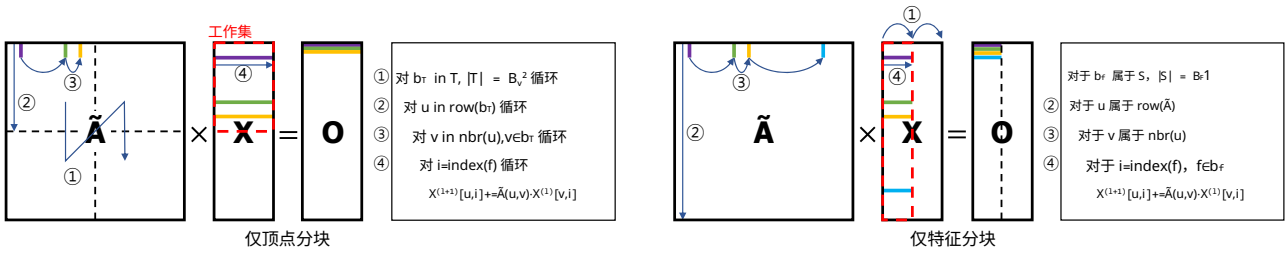
因此，直观而言，如果拓扑数据较小，增大 $|B_F|$ 会降低成本；如果特征矩阵较小，增大 $|B_V|$ 则更有利。由于图卷积网络中的特征宽度通常可达数百，因此通常优先增大$|B_F|$ 而非 $|B_V|$更有益。考虑两种极端情况：如图 8（左）所示，仅对拓扑结构进行分块（$|B_F| = 1$，$|B_V| = B$）；如图 8（右）所示，仅对特征矩阵进行分块（$|B_F| = B$，$|B_V| = 1$），使得工作集大小相等。将这些值代入公式 3，并求出前者小于后者的条件：

$$\begin{aligned} B|V| + |E| + B|V||F| &> B(|V| + |E|) + |V||F|, \\ |F| &> |E|/|V|. \end{aligned} \tag{4}$$

在上式中，$|E|/|V|$ 是图的平均度，是图数据的代表性参数之一。众所周知，现实世界图的平均度通常只有 10-20 [27]左右。与之相比，$|F|$ 可达数百，因此优先增大 $|B_F|$ 是合理的选择。

5 多芯片扩展

图 9 (a) 展示了多芯片环境。在多芯片扩展中，我们假设每个芯片都配备独立的内存模块，例如高带宽内存。在这种环境下，扩展挑战在于必须显式进行通信，因为各芯片的内存并不共享。图 9 (b) 和 (c) 展示了扩展策略。对于组合 (b)，我们将 W复制到每个芯片的内存中。由于 W很小，这种做法是可行的。随后，将 X沿水平方向拆分，使每个芯片都能生成输出矩阵的一部分。



Figu图 8：仅顶点分块（左）和仅特征分块（右）的两种极端循环分块情况 **t).**

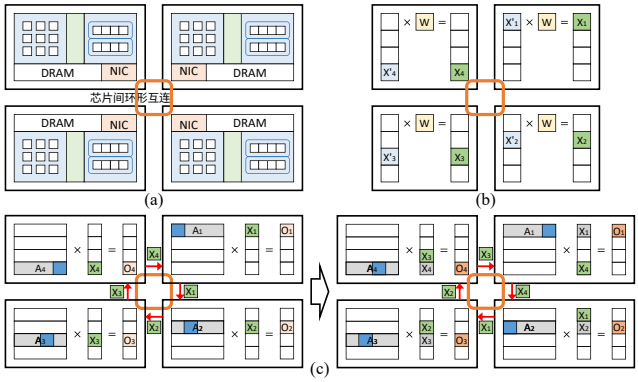


图 9：多芯片扩展策略。

表1：基准数据集信息

数据集	#顶点	#边	#特征	拓扑结构	特征
Products（PD）	2.45 M	61.9 M	100	0.24 GB	0.91 GB
Citation（CT）	2.93 M	30.6 M	128	0.12 GB	2.79 GB
Pokec（PK）	1.63 M	30.6 M	256	0.12 GB	1.56 GB
YouTube（YT）	1.13 M	2.99 M	256	0.02 GB	1.08 GB
LiveJournal（LJ）	4.85 M	69.0 M	256	0.28 GB	4.63 GB
Orkut（OK）	3.07 M	117 M	256	0.45 GB	2.93 GB
Reddit（RD）	0.23 M	114 M	602	0.43 GB	0.52 GB

˜对于聚合，拓扑结构 A 也会被拆分为四部分，无需在芯片之间通信。不过，X 需要进行通信，因为每个芯片仅拥有 X 的一部分，但最终需要整个矩阵。在我们的策略中，如图 9 (c) 所示，与全收集类似，[68], 每个芯片首先仅与 X当前部分进行聚合。同时，这一部分会传递给环形结构中的相邻芯片。接下来，刚接收到的数据部分会被处理并传递给下一个相邻芯片。经过 N个芯片上的 $N-1$ 步后，聚合便完成了。请注意，聚合之后不再进行通信（最后一个图卷积网络层除外），因为拆分后的输出会直接作为下一层组合的拆分输入。

6 评估

6.1 数据集

我们使用七个来自真实世界的大型图数据集作为基准。详细信息列于表1。Products（*PD*）来自开放图基准数据集 [27]。它包含 245 万个顶点，代表亚马逊商品，以及 6190 万条边，表示商品被共同购买。每个顶点都具有宽度为 100 的词嵌入作为特征，这些嵌入由商品描述生成。Citations（*CT*）也来自开放图基准数据集，其中微软学术图谱 [71] 中的 293 万篇论文作为顶点，论文之间的引用关系则构成 3060 万条边。每个顶点都关联着从论文标题和摘要生成的词嵌入。Pokec（*PK*）、YouTube（*YT*）、LiveJournal（*LJ*）、Orkut（*OK*）和 Reddit（*RD*）均来自斯坦福大型网络数据集合集 [42]。Pokec [67] 是一张来自斯洛伐克的社交网络图，包含 163 万个用户顶点和 3060 万条连接边。YouTube [77] 是由视频分享服务用户构成的图，包含 113 万个用户，以及他们之间的 299 万条连接。LiveJournal [43] 和 Orkut 来自在线社交网络服务，分别包含 485 万和 307 万个顶点。Reddit [76] 是一项社交网络服务，其中从 Reddit讨论论坛收集了 23 万个顶点。Reddit 数据集的特征包括 302 维的帖子信息，以及基于 GloVe CommonCrawl [56] 的 300 维词嵌入向量。对于其他数据集，宽度为 256 的节点嵌入向量由 node2vec [20] 生成，并用作特征。

6.2 方法

我们根据表2所列的配置评估 SnF 架构的性能。该加速器运行频率为 1GHz，其中组合引擎配备了 32×32 脉动阵列，聚合引擎配备了 16 路单指令多数据核心。两个引擎均采用 32 位定点运算。

我们采用 HyGCN 作为基准架构，因为 HyGCN 的架构易于扩展，以支持顶点分块和特征切片。另一方面，GCNAX 通过预先分析工作负载来进行静态顶点分块，而 AWB-GCN 不使用缓存来利用局部性。因此，在这类架构上应用采用动态策略和缓存的 SnF 并不可行。

我们在总共三种不同的设置下进行了实验。在默认设置中，我们采用多引擎配置，其中两种引擎各有八个，并配备 HBM2 内存子系统。在小规模设置中，我们展示了 SnF 在资源预算有限、内存带宽较低的设计上同样高效。该设置包含两种引擎各一个，并配备一个 4-rank DDR4 通道。最后，我们展示了扩展至第 5 节所述多芯片扩展方法的效果。我们使用 1 至 32 个芯片，并将它们连接成环形拓扑网络。与其他多芯片模块加速器类似 [5, 63], 我们采用带宽为 256GB/s、每跳延迟为 20ns 的带宽环形互连，每个芯片均连接到专属的 HBM 模块。为了

表2：系统配置

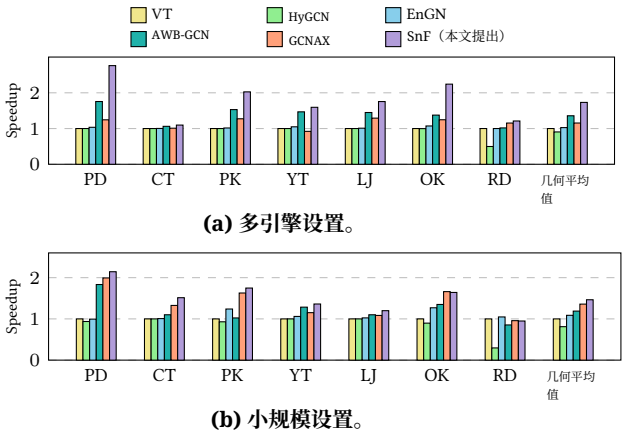
通用	加速器引擎	频率	1GHz
		组合	32×32 系统阵列
		聚合	16 路单指令多数据
	全局缓存	容量	16MB
		Ways	16
		替换	LRU
多引擎	#引擎	聚合	8
		组合	8
	片外内存（共享）	规格	HBM2
		峰值带宽	256 GB/s
		通道	8
		存储体	4×4
小规模	#引擎	聚合	1
		组合	1
	片外内存	规格	DDR4-2666
		峰值带宽	21.3 GB/s
		存储器层	4
		存储体	4×4
多芯片模块	模块数量	加速器	1-32
		HBM	1-32
	互连	拓扑结构	Ring
		带宽	256 GB/s
		延迟	20 纳秒

表 3：现有 GCN 加速器的比较

方案	数据流	循环分块		分块配置	
		拓扑结构	特征	方法	非完美分块
HyGCN	行乘积	✗	✗	✗	N/A
EnGN	行乘积	✗	✓	✗	N/A
AWB-GCN	列乘积	✓	✓	✗	N/A
GCNAX	行乘积	✓	✓	预处理	✗
SnF（提出的方法）	行乘积	✓	✓	ATM（运行时）	✓

验证该设计，我们使用 Verilog 硬件描述语言实现了加速器，并使用 Cadence Genus 和 45 纳米 OpenPDK 库对其进行综合，再将其缩放至 32 纳米。为了对片上内存进行建模，我们使用 CACTI [69]，在 32 纳米工艺节点下估算功耗和面积。与 HyGCN [75] 相似的基线芯片功耗为 11.40 mm^2, ，而 SnF 的面积为 11.55 mm^2。作为对比，我们复现了 GCNAX [46]和 AWB-GCN [17]„ 其面积分别为 11.23 mm^2 和 13.80 mm^2, 。

在性能建模方面，我们将 SCALE-Sim [61] 移植到 C++，并扩展其功能以支持 SnF 的聚合单元。其周期数与 RTL 模型相匹配。对于 DRAM 子系统建模，我们使用了 DRAMsim 3 [49]。我们忠实地复现了表 3 中的先前工作。HyGCN [75] 是采用混合引擎的基线加速器；EnGN [50] 提出了结合 DAVC 的 RER 数据流，用于处理高阶顶点。AWB-GCN [17] 基于列乘积，使用同一组处理单元进行聚合和组合，并配备独立的任务分配器。最后，GCNAX [46] 通过静态分析，在完美分块的假设下寻找最优循环分块。为便于比较，我们还加入了加速器的“VT”版本，并以此对所有数值进行归一化。



(a) 多引擎设置。

(b) 小规模设置。

图10：SnF 在 (a) 多引擎设置和 (b) 小规模设置下的性能。

6.3 多引擎加速器结果

在本小节中，我们分析多引擎性能，以展示所提出的 SnF 优于现有加速器。我们在加速器中部署了八个引擎，并由 HBM2 内存提供 256 GB/s 的峰值带宽。结果如图 10a 所示。我们将“VT”设为基线，它仅执行顶点分块。“SnF”是所提出的加速器，在顶点分块的基础上进行特征切片，并采用自动瓦片变形。在多个引擎之间，我们对图工作负载进行划分，使每个引擎处理的边数相等。除动态划分的 SnF 外，对于所有实现，我们都离线检查了所有可能的分区组合（即最优组合），并选取其中最佳方案。

将 SnF 与 VT 进行比较，可以深入了解为图卷积网络推理选择合适的循环分块方案能够带来多大收益。SnF 相较于 VT 的性能提升为 73.1%，这是通过减少 VT 重复特征访问开销所造成的大量开销，并动态寻找合适的配置实现的。最大的加速比出现在 PD（2.76×）上，这是因为 VT 的最优设置为 large $B_V = 64$。这个较大的 B_V 导致输出特征访问产生了巨大的重复开销。特征切片能够有效拆分这些重复次数，从而带来更高的加速比。

所提出的技术在所有数据集上的表现均优于现有设计中的循环分块策略。HyGCN 不采用分块，因此缓存效率较低。EnGN 基于顶点分块，并使用 DAVC 捕获高阶顶点的特征。然而，我们发现 DAVC 对加速比的影响微乎其微。原因之一是，在大型真实世界图数据集中，高阶顶点数量过多，DAVC 无法将其全部捕获。此外，当这些顶点的重用距离足够短时，默认的最近最少使用替换策略通常已经能够很好地捕获它们。在受测设计中，AWB-GCN 独树一帜，采用基于列乘积的执行方式并进行积极的负载均衡。由于图中有数百万个顶点，若不单独考虑溢出问题，就需要一个不可忽视的大型输出缓冲区才能运行。尽管 AWB-GCN 对矩阵进行了划分，却增加了特征访问的重复次数，

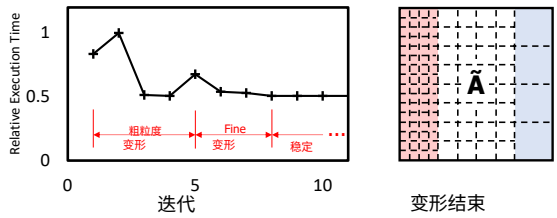


图11：跨多次迭代进行自动瓦片变形的示例。

因此其表现比 SnF 低 35.7%。GCNAX 提供了一种解析模型，用于寻找最优划分方案；与 VT 相比，其加速比达到 15.4%。然而，该模型仅考虑完美分块，错失了一个重要的性能提升机会。此外，由于它不切实际地假设稀疏图均匀分布，因此模型结果往往与实际测量值相去甚远，这也导致其性能比 SnF 低 68.2%。

图11展示了在PK数据集的多次重复执行中，自动瓦片变形的表现。前四次迭代采用粗粒度变形。该方法逐步检查多个 B_V 值，并最终确定为 $B_V = 8$。这些非最优执行带来的开销会分摊到多次迭代中。在精细变形阶段，它最终进一步切分了命中率最低的第一列，并合并了命中率最高的最后两列。在我们的实验中，自动瓦片变形能够达到最优分析所能实现性能的95%以上，而运行时无法获取最优分析所需的信息。

6.4 小规模加速器结果

在本小节中，我们分析单引擎加速器的性能，以说明特征切片和自动瓦片变形对于资源预算紧张的系统（如移动设备和嵌入式系统）同样有效。图10b展示了图卷积网络执行的小规模性能。

在多引擎设置中，SnF 优于此前所有设计。一个有趣的现象是，在此前的设计中，GCNAX 在小规模设置下的表现优于 AWB-GCN。在这种设置下，可用内存带宽相当有限，因此高效利用缓存至关重要。由于 AWB-GCN 没有专门的缓存机制，高效利用缓存所带来的收益比负载均衡更为重要。除 OK 和 RD 外，SnF 在所有数据集上都能找到更优的设置；与 GCNAX 相比提升了 10.8%，与 VT 相比提升了 46.4%。

与其他数据集相比，Reddit 的每个顶点边数最多（拓扑结构密集），但顶点数较少，因此其性能加速比呈现出独特的趋势。如图10所示，RD 的密集拓扑结构与 VT 配合良好，VT 的加速比优于 HyGCN。此外，顶点较少意味着其特征矩阵小于其他数据集的特征矩阵，从而降低了 VT 的输出写入开销。GCNAX 使用完美分块，因此构建的 B_V 分块策略过大，导致其加速比低于 VT。对于 SnF，自动瓦片变形找到的动态分块几乎与 GCNAX 找到的静态方案相同。由于自动瓦片变形需要经过若干次迭代才能稳定在最佳方案附近

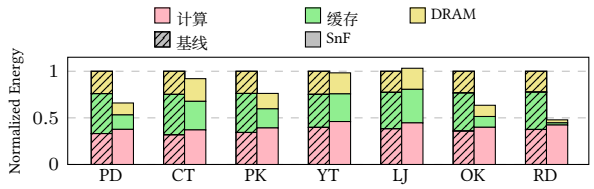


图12：基线（阴影部分）和 SnF（实色部分）的归一化能耗细分。

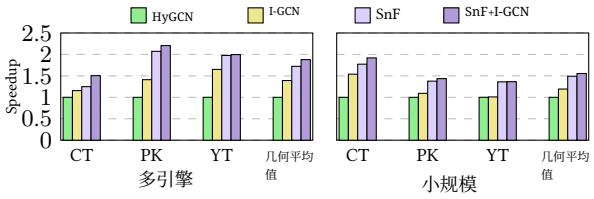


图13：I-GCN 重排序的性能。

如图10b所示，SnF 的加速比因此比 GCNAX 略低 0.82%。OK 的拓扑结构也与 Reddit 一样密集，但其顶点数较多。这种情况不利于 VT。因此，与 Reddit 的情况不同，GCNAX 和 SnF 的表现都优于 VT。与 Reddit 的情况类似，SnF 的加速比比 GCNAX 略低 1.8%。

6.5 能耗

图12展示了单芯片、多引擎设置下聚合阶段的归一化能耗及其细分情况。我们将未进行分区的架构设为基线，其能耗以阴影柱表示。SnF 的能耗则以实心柱表示。如图所示，SnF 的计算能耗略高，但从内存和缓存中节省了更多能耗。这两者都表明 SnF 高效地利用了缓存：由于缓存命中率提高，从动态随机存取存储器读取的数据更少，写入缓存的数据也更少。LJ 是个例外，其从缓存和动态随机存取存储器中节省的能耗略低于计算能耗的增幅。我们在 RD 上节省了 53% 的能耗，因为它具有极为密集的拓扑结构，且相对于特征访问，其拓扑访问比更小。因此，由于 SnF 能显著减少特征访问，能耗也大幅降低。

6.6 重排序的影响

Rubik [11] 和 I-GCN [18]采用的图重排序，是另一种提升图卷积网络缓存效率的有效方案。SnF 能够从这些重排序方案中进一步获益，原因有二。首先，图经过重排序后会形成小而密集的簇。因此，访问模式会变得类似于较小工作集的访问模式。这会改变顶点分块的最优数量，而 SnF 借助自动瓦片变形能够适应这种变化。其次，重排序有助于改善访问的局部性。SnF 利用缓存来发挥访问局部性的优势，因此重排序通常会增强 SnF 的效果。

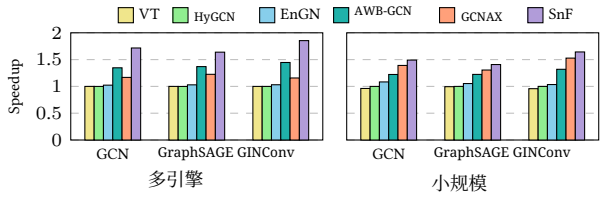


图 14：不同图神经网络应用的性能。

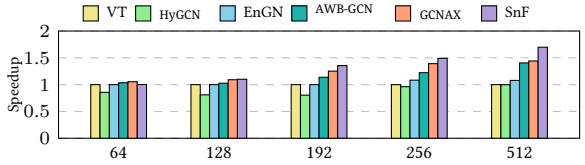


图 15：SnF 在不同特征宽度下的性能。

图 13 展示了在三个数据集上，结合 I-GCN [18]重排序实现的 SnF 所带来的性能提升。如图所示，将重排序与 SnF 结合，始终比单独使用任一方案带来额外加速比。在多引擎设置下，重排序与 SnF 的组合相较于 I-GCN 的几何平均性能提升了 87.9%，相较于原始 SnF 的几何平均性能提升了 4.22%。在小规模设置下，相较于 I-GCN 的性能提升为 55.4%，相较于原始 SnF 的性能提升为 9.04%。

6.7 在图卷积网络变体上的性能

尽管 SnF 专注于加速图卷积网络，但图卷积网络存在许多变体 [23, 74]，SnF 也可以用于执行这些变体。我们实现了两种流行的图卷积网络变体 GraphSAGE [23] 和 GINConv [74]，，它们对聚合阶段进行了轻微修改。图 14 绘制了图卷积网络的性能。由于它们采用相同的聚合-组合结构，每项结果都呈现出几乎相同的趋势。在 GraphSAGE 上，SnF 在多引擎设置下相较于 VT 获得了 63.9% 的加速比，在小规模设置下获得了 40.7% 的加速比。另一方面，在 GINConv 上，SnF 在多引擎设置下获得了 85.4% 的加速比，在小规模设置下获得了 64.3% 的加速比。由于 GraphSAGE 采用采样，尽量减少了边访问次数，从而降低了特征复用的机会，因此其性能增益略低于图卷积网络。另一方面，GINConv 使用了更小规模的边权重集合$\tilde{}$（即 A的一部分），因此读取特征带来的加速比更加明显，最终其加速比高于图卷积网络。

6.8 对特征宽度的敏感性

在本小节中，我们对 SnF 进行了敏感性分析。根据公式 4 的一项洞见，特征切片的加速比取决于 B_F。图 15 展示了特征宽度对性能的影响敏感性，结果以几何平均值表示。如图所示，对于特征宽度较小的层，SnF 的效果会有所降低，而这类层通常出现在图卷积网络的后几层。不过，由于这些层的执行时间较短，因此其影响微乎其微。此外，SnF 找到的配置可以在后续层中重复使用。随着特征宽度增大、层数加深，AT M 的收益可能会逐渐提升。

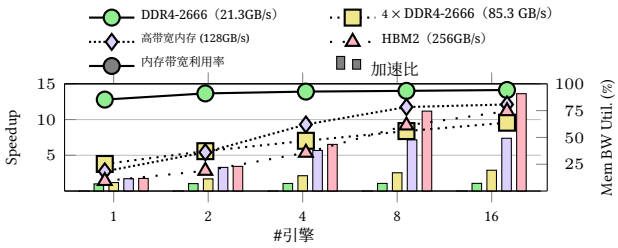


图16：SnF 对不同内存系统和引擎数量的敏感性。

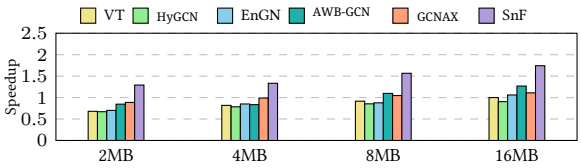


图 17：不同缓存大小下 SnF 的性能（几何平均值）。

6.9 对系统的敏感性

图 16 展示了 SnF 在不同引擎数量和不同内存系统下的性能，并以配备一个 DDR4-2666 通道的小规模 SnF 的性能为基准进行归一化。一个 DDR4-2666 通道的峰值带宽为 21.3GB/s；当执行受内存带宽限制时，增加引擎数量带来的收益会逐渐减少。然而，内存子系统的带宽越高，就需要越多引擎才能充分利用带宽。例如，带宽为 256GB/s 的 HBM2 系统可以从增加处理引擎中获益，并在引擎数量达到八个之前实现近乎线性的加速。此后，性能便达到饱和。根据这项研究，我们将默认配置中的引擎数量设为八个。

6.10 对缓存大小的敏感性

分析缓存大小敏感性至关重要，因为 SnF 利用了高效的缓存局部性。图 17 展示了 SnF 在不同缓存大小下的性能。SnF 的整体表现优于其他架构。在缓存较小的情况下，SnF 尤为占优，因此可以采用较小的缓存来减少硅面积。在小缓存的情况下，SnF 致力于降低的特征访问未命中率会上升，因此相较于其他加速器，SnF 能带来更高的加速比。例如，在 2MB 配置下，SnF 的性能比 VT 提升了 96.7%；而在 16 MB 配置下，提升幅度为 73.1%，仍然是显著的加速比。

6.11 对缓存配置的敏感性

图 18 展示了对各种缓存配置的敏感性，包括替换策略、路数和块大小。在替换策略方面，我们测试了随机替换和 RRIP [84]。随机替换导致加速比出乎意料地大幅下降，因为它无法捕捉局部性。RRIP 带来了少量加速比提升，但不足以证明存储更多状态所需的额外开销是合理的。路数的变化并未带来

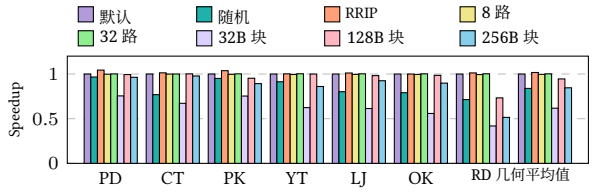


图 18：对不同缓存配置的敏感性。

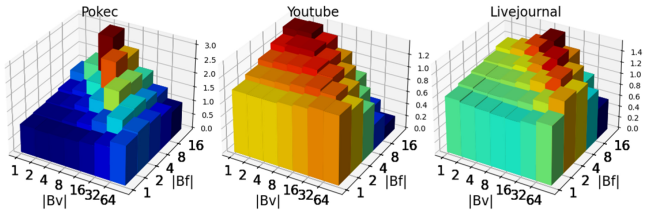


图 19：对 B_F 和 B_V 的敏感性。

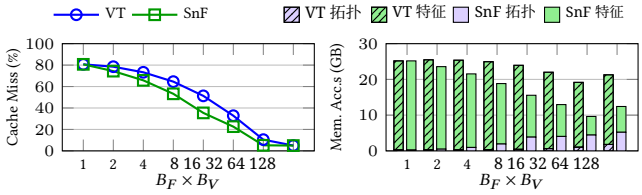


图 20：根据 $B_F \times B_V$ 得出的未命中率（左）和内存访问量（右）。

太大影响，而块大小则显著影响了性能。块大小减小时，会小于 DRAM 突发传输的大小，浪费大量带宽。块大小增大时，则会超过特征切片有限的空间局部性，导致加速比下降。

6.12 对 B_F 和 B_V 的敏感性

图 19 展示了 SnF 在三个选定数据集上对 B_F 和 B_V 的敏感性。结果体现了调整顶点分块数量 B_V 的难度。最优配置通常出现在 B_F=16 （最大值）处，这支持了我们的策略：先进行特征切片，再自适应地寻找最优的B_V。另一方面， B_V 较难调整，因为它过于依赖具体的图拓扑。较大的图倾向于使用更大的 B_V，因为较大的图更难装入缓存。

图 20（左）展示了 PK 数据集在 $B_F \times B_V$ 下观测到的缓存未命中率。随着 $B_F \times B_V$ 增大，工作集大小减小，未命中率也随之降低。仅 VT 与 SnF 的未命中率降幅大致相似，这表明我们的成本模型假设总体成立：未命中率大致是工作集大小的函数。

图 20（右）展示了内存访问次数的细分情况，说明未命中率和重复次数如何转化为性能。绿色条表示特征访问，紫色条表示拓扑数据访问。在顶点分块中，输出特征访问增加的重复次数抵消了未命中率下降带来的收益，导致收益递减。然而，对于 SnF，未命中率的下降几乎直接

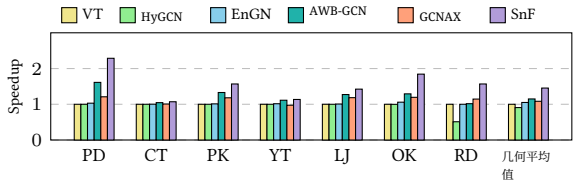


图 21：多芯片加速器的性能。

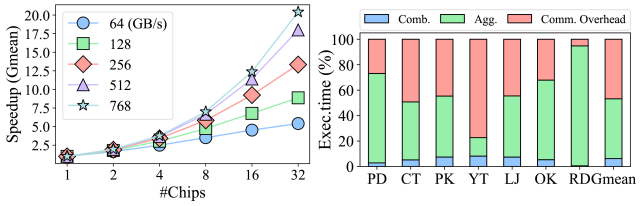


图 22：多芯片扩展结果（左）和执行时间分解（右）。

转化为内存访问减少，因为重复操作是针对规模较小的拓扑数据增加的。

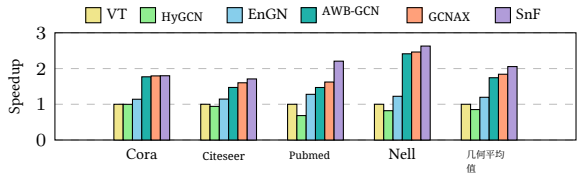
6.13 多芯片模块扩展结果

图 21 展示了多芯片模块（MCM）加速器的性能，其中总共 16 个芯片组成一个模块。与其他多芯片模块加速器一样，[5, 63] 我们采用带宽为 256 GB/s、每跳延迟为 20ns 的环形互连。尽管所有加速器都采用了第 5 节所述的相同策略，SnF 的加速比仍优于其他所有方案。图 22 展示了更多扩展结果。左图根据互连带宽绘制了可扩展性。使用高带宽互连时，系统可以在一定程度上扩展；但对于低带宽互连，加速比则受到限制。右图展示了组合、聚合以及未被重叠隐藏的通信开销的分解情况。

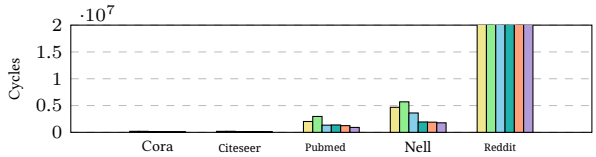
PD、OK 和 RD 等平均度较大的数据扩展效果良好，因为通信时间取决于每个顶点的特征数，而计算时间通常与边数成正比。另一方面，YT 等低度图的可扩展性较差，这是由于其平均度较低，因而通信开销较高。

6.14 小规模数据集上的实验

除了我们评估的数据集外，先前的一些研究也经常评估一些小规模数据集 [17, 46, 75]：Cora、Citeseer [62], Pubmed [22], 和 Nell [9]。为进行公平比较，我们也在图23中报告了这些数据集上的性能。按照 [46] 的做法，我们为所有架构采用了 512 KB 的缓存大小。SnF 相较于基线的性能提升为 106%，远高于 SnF 在大型数据集上取得的结果，如图23a所示。这些结果与先前研究 [17, 46] 的观察一致。不过，我们没有在主要评估中使用这些小规模数据集，因为它们通常仅包含数千个顶点，因此与较大的数据集相比，计算图卷积网络所需的周期极少，如图23b所示。



（a）小规模数据集上的加速比。



（b）处理图卷积网络计算所需的周期数。

图23：小规模数据集上的结果。

7 相关工作

7.1 图卷积网络与图处理加速器

现有许多旨在加速图卷积网络的架构方法。HyGCN [75] 利用图卷积网络中聚合阶段和组合阶段的特性差异，提出了混合计算架构。GNNA [8] 使用分块架构加速图卷积网络。EnGN [50] 采用分块拓扑数据，并收集高频访问顶点的信息，为其分配部分缓存，以提高缓存效率。此外，GCNAX [46] 通过离线分析采用完美分块策略，以找到最佳数据流和分块配置。然而，其完美分块高估了所需的缓冲区大小，导致性能欠佳。相比之下，本文专注于缓存效率，并指出特征切片通常比先前的解决方案效率更高。AWB-GCN [17] 使用基于列的执行，并提出了工作负载分配方面的见解。近期研究还支持将边信息或自注意力融入图卷积网络的变体。对于此类图卷积网络变体，SnF 也展现出类似的加速比，因为特征切片和动态分析技术很容易应用，且无需克服太多困难。

图卷积网络之外，经典图处理数十年来一直是重要的应用领域，也有许多工作提出了专用加速器。这些加速器采用专用集成电路 [21, 53, 80], FPGA [7, 13, 40], 或二者兼用 [10] 来实现。针对图处理的内存密集型特性，有些加速器利用 3D 堆叠内存 [55] 作为存内处理加速器 [1, 82, 90], ，也有些通过改造闪存存储，使其更适用于图处理。

遗憾的是，这类经典图处理的计算模式与图卷积网络的计算模式截然不同，因为每个顶点关联的特征通常很浅，每个顶点只有一两个元素。由于图卷积网络具有混合计算模式，且每个顶点关联的特征向量相对较长，因此图卷积网络加速器需要采用不同于图处理加速器的设计。

7.2 面向内存感知的深度神经网络加速器

深度神经网络加速通常伴随着内存受限操作，因此许多研究通过优化内存带宽来实现有效加速比 [31, 59, 65]。一些研究会精准选择数据流 [12, 15, 34] 或数据块 [78] ，以提高数据复用率。片上内存管理对优化也至关重要。许多研究尝试通过调度、分配或共享 [33, 47, 73]，高效管理片上内存。深度神经网络通常包含多个层，可利用这一特点优化内存使用 [4, 30, 36, 51]。这一特点不仅可应用于推理 [4, 30]和训练 [51]，也可应用于批归一化 [36]。

这些方法适用于传统深度神经网络的规则访问模式，但并不直接适用于图卷积网络。因此，图卷积网络加速器的设计不同于传统深度神经网络加速器的设计，因为其针对的是不规则访问。

7.3 稀疏深度神经网络加速器

一些深度神经网络加速器尝试利用深度神经网络中的稀疏性。计算剪枝 [28, 52, 64] 曾是一个热门目标，因为它可以轻松应用于现有硬件设计。稀疏操作支持 [72, 88] 或代数稀疏性探索 [29] 也得到了研究。此外，还有零值感知设计，例如移除零值权重 [3, 14, 24, 83, 87] 或考虑零值激活 [19, 32, 35, 54]。一些设计针对 SpMV [6, 60] 和 SpMM [25, 58, 85], ，可用于利用深度神经网络中的稀疏性。另外，一些工作 [37, 57, 66, 79] 对软件和硬件进行了协同设计，以实现张量压缩。然而，这些工作所针对的稀疏性通常远低于图卷积网络中的稀疏性，因为图拓扑的稀疏性接近 100%，这是经过剪枝的深度神经网络或零值激活难以达到的。因此，尽管针对稀疏深度神经网络加速已经开展了大量工作，但仍然需要专用的图卷积网络加速器。

8 结论

我们提出了切片与锻造，这是一种面向 GCN 加速器、旨在高效利用缓存的加速器设计。与以往工作相比，切片与锻造通过特征切片大幅提升了有限片上缓存的利用效果。利用特征切片会产生多个完全相同的计算模式这一特点，我们进一步提出了自动瓦片变形技术，可动态配置分块以提升缓存效率。实验结果表明，切片与锻造的性能优于现有设计，并降低了对离线分析的需求。

致谢

本研究部分ได้รับ韩国国家研究基金会（NRF）资助，项目编号为（2022R1C1C1011307、2022R1C1C1008131）；同时还获得韩国政府（韩国科学技术信息通信部，MSIT）资助的信息通信技术规划评估院（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、Hyeyoon Lee、Jounghoo Lee和Youngsok Kim任职于延世大学计算机科学系，并部分获得韩国教育部（MOE）与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] Junwhan Ahn、Sungpack Hong、Sungjoo Yoo、Onur Mutlu和Kiyoung Choi。2015年。一种用于并行图处理的可扩展存内处理加速器。载于ISCA。[2] Hasan Metin Aktulga、Aydin Buluç、Samuel Williams和Chao Yang。2014年。针对核组态相互作用计算优化稀疏矩阵-多向量乘法。载于IPDPS。[3] Jorge Albericio、Patrick Judd、Tayler Hetherington、Tor Aamodt、Natalie Enright Jerger和Andreas Moshovos。2016年。Cnvlutin：无无效神经元的深度神经网络计算。载于ISCA。[4] Manoj Alwani、Han Chen、Michael Ferdman和Peter Milder。2016年。融合层CNN加速器。载于MICRO。[5] Akhil Arunkumar、Evgeny Bolotin、Benjamin Cho、Ugljesa Milic、Eiman Ebrahimi、Oreste Villa、Aamer Jaleel、Carole-Jean Wu和David Nellans。2017年。MCM-GPU：用于持续提升性能可扩展性的多芯片模块GPU。MICRO 45，2（2017），320–332。[6] Bahar Asgari、Ramyad Hadidi、Tushar Krishna、Hyesoon Kim和Sudhakar Yalamanchili。2020年。ALRESCHA：轻量级可重构稀疏计算加速器。载于HPCA。[7] Mikhail Asiatici和Paolo Ienne。2021年。借助缓存处理数千个并发未命中请求的FPGA大规模图处理。载于ISCA。[8] Adam Auten、Matthew Tomei和Rakesh Kumar。2020年。图神经网络的硬件加速。载于DAC。[9] Andrew Carlson、Justin Betteridge、Bryan Kisiel、Burr Settles、Estevam R. Hruschka和Tom M. Mitchell。2010年。迈向永不停歇的语言学习架构。载于第二十四届AAAI人工智能会议（美国佐治亚州亚特兰大）（AAAI'10）。AAAI出版社，1306–1313。[10] Xuhao Chen、Tianhao Huang、Shuotao Xu、Thomas Bourgeat、Chanwoo Chung和Arvind。2021年。FlexMiner：一种模式感知图模式挖掘加速器。载于ISCA。[11] Xiaobing Chen、Yuke Wang、Xinfeng Xie、Xing Hu、Abanti Basak、Ling Liang、Mingyu Yan、Lei Deng、Yufei Ding、Zidong Du、Yunji Chen和Yuan Xie。2020年。Rubik：一种用于高效图学习的分层架构。CoRR abs/2009 12495 2020 2009 12495 2009 12495 。（）。arXiv：。[12] 2016 https://arxiv.org/abs/。Yu-Hsin Chen、Tushar Krishna、Joel S Emer和Vivienne Sze。。Eyeriss：一种用于深度卷积神经网络的高能效可重构加速器。JSSC（2016）。[13] Vidushi Dadu、Sihao Liu和Tony Nowatzki。2021年。PolyGraph：展现灵活性对图处理加速器的价值。载于ISCA。[14] Caiwen Ding、Siyu Liao、Yanzhi Wang、Zhe Li、Ning Liu、Youwei Zhuo、Chao Wang、Xuehai Qian、Yu Bai、Geng Yuan、Xiaolong Ma、Yipeng Zhang、Jian Tang、Qinru Qiu、Xue Lin和Bo Yuan。2017年。CirCNN：使用分块循环权重矩阵加速并压缩深度神经网络。载于MICRO。[15] Zidong Du、Robert Fasthuber、Tianshi Chen、Paolo Ienne、Ling Li、Tao Luo、Xiaobing Feng、Yunji Chen和Olivier Temam。2015年。ShiDianNao：将视觉处理移至更靠近传感器的位置。载于ISCA。[16] Alex Fout、Jonathon Byrd、Basir Shariat和Asa Ben-Hur。2017年。使用图卷积网络进行蛋白质界面预测。载于NeurIPS。[17] Tong Geng、Ang Li、Runbin Shi、Chunshu Wu、Tianqi Wang、Yanfei Li、Pouya Haghi、Antonino Tumeo、Shuai Che、Steve Reinhardt和Martin C. Herbordt。2020年。AWB-GCN：一种具备运行时工作负载再平衡功能的图卷积网络加速器。载于MICRO。[18] Tong Geng、Chunshu Wu、Yongan Zhang、Cheng Tan、Chenhao Xie、Haoran You、Martin Herbordt、Yingyan Lin和Ang Li。2021年。I-GCN：一种通过岛化实现运行时局部性增强的图卷积网络加速器。载于MICRO。美国计算机协会，纽约，纽约州，美国，1051–1063。https://doi.org/ 10 1145 3466752 3480113 。/。[19] Ashish Gondimalla、Noah Chesnut、Mithuna Thottethodi和T. N. Vijaykumar。2019年。SparTen：一种用于卷积神经网络的稀疏张量加速器。载于MICRO。[20] Aditya Grover和Jure Leskovec。2016年。node2vec：面向网络的可扩展特征学习。载于KDD。[21] Tae Jun Ham、Lisa Wu、Narayanan Sundaram、Nadathur Satish和Margaret Martonosi。2016年。Graphicionado：一种高性能、高能效的图分析加速器。载于MICRO。[22] Will Hamilton、Zhitao Ying和Jure Leskovec。2017年。大型图上的归纳式表示学习。载于《神经信息处理系统进展》，I. Guyon、U. Von Luxburg、S. Bengio、H. Wallach、R. Fergus、S. Vishwanathan和R. Garnett（编），第30卷。Curran Associates, Inc.
h
t
t
p
s://
p
r
o
ceedings.neurips.cc/paper/2017/file/5dd9db5e033da9c6fb5ba83c7a7ebea9-Paper.pdf
[23] William L Hamilton、Rex Ying和Jure Leskovec。2017年。大型图上的归纳式表示学习。载于NeurIPS。[24] Song Han、Xingyu Liu、Huizi Mao、Jing Pu、Ardavan Pedram、Mark A Horowitz和William J Dally。2016年。EIE：压缩深度神经网络上的高效推理引擎。载于ISCA。[25] Kartik Hegde、Hadi Asghari-Moghaddam、Michael Pellauer、Neal Crago、Aamer Jaleel、Edgar Solomonik、Joel Emer和Christopher W Fletcher。2019年。ExTensor：一种用于稀疏张量代数的加速器。发表于 MICRO。[26] Changwan Hong、Aravind Sukumaran-Rajam、Israt Nisa、Kunal Singh 和 P Sadayappan。2019年。用于稀疏矩阵乘法的自适应稀疏分块。发表于 PPoPP。[27] Weihua Hu、Matthias Fey、Marinka Zitnik、Yuxiao Dong、Hongyu Ren、Bowen Liu、Michele Catasta 和 Jure Leskovec。2020年。开放图基准：用于图机器学习的数据集。arXiv 预印本（2020年）。[28] Weizhe Hua、Yuan Zhou、Christopher De Sa、Zhiru Zhang 和 G. Edward Suh。2019年。利用动态细粒度通道门控提升 CNN 加速器的性能。发表于 MICRO。[29] Chao-Tsung Huang。2021年。RingCNN：利用代数稀疏环形张量实现节能型基于CNN的计算成像。发表于 ISCA。[30] Chao-Tsung Huang、Yu-Chun Ding、Huan-Ching Wang、Chi-Wen Weng、Kai-Ping Lin、Li-Wei Wang 和 Li-De Chen。2019年。eCNN：一种面向边缘推理、基于块且高度并行的 CNN 加速器。发表于 MICRO。[31] Bongjoon Hyun、Youngeun Kwon、Yujeong Choi、John Kim 和 Minsoo Rhu。2020年。NeuMMU：为 NPU 中高效地址转换提供架构支持。发表于 ASPLOS。[32] Jun-Woo Jang、Sehwan Lee、Dongyoung Kim、Hyunsun Park、Ali Shafiee Ardestani、Yeongjae Choi、Channoh Kim、Yoojin Kim、Hyeongseok Yu、Hamzah Abdel-Aziz、Jun-Seok Park、Heonsoo Lee、Dongwoo Lee、Myeong Woo Kim、Han-woong Jung、Heewoo Nam、Dongguen Lim、Seungwon Lee、Joon-Ho Song、Suk-nam Kwon、Joseph Hassoun、SukHwan Lim 和 Changkyu Choi。2021年。面向三星旗舰移动 SoC 的稀疏性感知、可重构 NPU 架构。发表于 ISCA。[33] Tian Jin 和 Seokin Hong。2019年。Split-CNN：拆分卷积神经网络中的基于窗口的操作以优化存储系统。发表于 ASPLOS。[34] Norman P. Jouppi、Cliff Young、Nishant Patil、David Patterson、Gaurav Agrawal、Raminder Bajwa、Sarah Bates、Suresh Bhatia、Nan Boden、Al Borchers、Rick Boyle、Pierre-luc Cantin、Clifford Chao、Chris Clark、Jeremy Coriell、Mike Daley、Matt Dau、Jeffrey Dean、Ben Gelb、Tara Vazir Ghaemmaghami、Rajendra Gottipati、William Gulland、Robert Hagmann、C. Richard Ho、Doug Hogberg、John Hu、Robert Hundt、Dan Hurt、Julian Ibarz、Aaron Jaffey、Alek Jaworski、Alexander Kaplan、Harshit Khaitan、Daniel Killebrew、Andy Koch、Naveen Kumar、Steve Lacy、James Laudon、James Law、Diemthu Le、Chris Leary、Zhuyuan Liu、Kyle Lucke、Alan Lundin、Gordon MacKean、Adriana Maggiore、Maire Mahony、Kieran Miller、Rahul Nagarajan、Ravi Narayanaswami、Ray Ni、Kathy Nix、Thomas Norrie、Mark Omernick、Narayana Penukonda、Andy Phelps、Jonathan Ross、Matt Ross、Amir Salek、Emad Samadiani、Chris Severn、Gregory Sizikov、Matthew Snelham、Jed Souter、Dan Steinberg、Andy Swing、Mercedes Tan、Gregory Thorson、Bo Tian、Horia Toma、Erick Tuttle、Vijay Vasudevan、Richard Walter、Walter Wang、Eric Wilcox 和 Doe Hyun Yoon。2017年。张量处理器的数据中心内性能分析。发表于 ISCA。[35] Patric Judd、Alberto Delmas、Sayeh Sharify 和 Andreas Moshovos。2017年。Cnvlutin²：无需无效激活和权重的深度神经网络计算。arXiv 预印本（2017年）。[36] Daejin Jung、Wonkyung Jung、Byeongho Kim、Sunjung Lee、Wonjong Rhee 和 Jung Ho Ahn。2019年。重构批归一化以加速 CNN 训练。发表于 SysML。[37] Konstantinos Kanellopoulos、Nandita Vijaykumar、Christina Giannoula、Roknoddin Azizi、Skanda Koppula、Nika Mansouri Ghiasi、Taha Shahroodi、Juan Gomez Luna 和 Onur Mutlu。2019年。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。[38] Kevin Kiningham、Christopher Re 和 Philip Levis。2020年。GRIP：一种图神经网络加速器架构。arXiv 预印本（2020年）。[39] Thomas N Kipf 和 Max Welling。2016年。利用图卷积网络进行半监督分类。arXiv 预印本（2016年）。[40] Jinho Lee、Heesu Kim、Sungjoo Yoo、Kiyoung Choi、H Peter Hofstee、Gi-Joon Nam、Mark R Nutter 和 Damir Jamsek。2017年。Extra V：利用缓存一致性加速器提升存储设备附近的图处理性能。pVLDB（2017年）。[41] Jure Leskovec、Jon Kleinberg 和 Christos Faloutsos。2005年。随时间演变的图：稠密化定律、直径缩小及可能的解释。发表于 KDD。[42] Jure Leskovec 和 Andrej Krevl。2014年。SNAP 数据集：斯坦福大型网络数据集合集。http://snap.stanford.edu/data。[43] Jure Leskovec、Kevin J Lang、Anirban Dasgupta 和 Michael W Mahoney。2009年。大型网络中的社区结构：自然簇大小与缺乏大型定义明确的簇。Internet Mathematics 6，1（2009年），29–123。[44] Guohao Li、Matthias Muller、Ali Thabet 和 Bernard Ghanem。2019年。DeepGCNs：图卷积网络能像卷积神经网络一样加深吗？发表于 ICCV。[45] Han Li、Mingyu Yan、Xiaocheng Yang、Lei Deng、Wenming Li、Xiaochun Ye、Dongrui Fan 和 Yuan Xie。2021年。通过双向融合加速图卷积网络。IEEE计算机体系结构快报（2021年）。[46] Jiajun Li、Ahmed Louri、Avinash Karanth 和 Razvan Bunescu。2021年。GCNAX：一种灵活且节能的图卷积神经网络加速器。发表于 HPCA。[47] Jiajun Li、Guihai Yan、Wenyan Lu、Shuhao Jiang、Shijun Gong、Jingya Wu 和 Xiaowei Li。2018年。SmartShuttle：优化深度

学习加速器的片外存储器访问。发表于 DATE。[48] Qimai Li、Zhichao Han 和 Xiao-Ming Wu。2018年。深入探究用于半监督学习的图卷积网络。发表于 AAAI。[49] Shang Li、Zhiyuan Yang、Dhiraj Reddy、Ankur Srivastava 和 Bruce Jacob。2020年。DRAMsim3：周期精确且支持热特性的动态随机存取存储器模拟器。CAL（2020年）。[50] Shengwen Liang、Ying Wang、Cheng Liu、Lei He、LI Huawei、Dawen Xu 和 Xiaowei Li。2020年。EnGN：面向大型图神经网络的高吞吐量、节能加速器。TC（2020年）。[51] Sangkug Lym、Armand Behroozi、Wei Wen、Ge Li、Yongkee Kwon 和 Mattan Erez。2019年。小批量序列化：通过层间数据复用进行 CNN 训练。发表于 SysML。[52] Wei Niu、Xiaolong Ma、Sheng Lin、Shihao Wang、Xuehai Qian、Xue Lin、Yanzhi Wang 和 Bin Ren。2020年。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 ASPLOS。[53] Muhammet Mustafa Ozdal、Serif Yesil、Taemin Kim、Andrey Ayupov、John Greth、Steven Burns 和 Ozcan Ozturk。2016年。图分析加速器的节能架构。发表于 ISCA。[54] Angshuman Parashar、Minsoo Rhu、Anurag Mukkara、Antonio Puglielli、Rangharajan Venkatesan、Brucek Khailany、Joel Emer、Stephen W. Keckler 和 William J. Dally。2017年。SCNN：一种用于压缩稀疏卷积神经网络的加速器。发表于 ISCA。[55] J Thomas Pawlowski。2011年。混合内存立方体（HMC）。发表于 Hot Chips。[56] Jeffrey Pennington、Richard Socher 和 Christopher D. Manning。2014年。GloVe：用于词表示的全局向量。发表于自然语言处理经验方法会议（EMNLP）。1532–1543。http://www.aclweb.org/anthology/D14-1162[57] Lillian Pentecost、Marco Donato、Brandon Reagen、Udit Gupta、Siming Ma、Gu-Yeon Wei 和 David Brooks。2019年。MaxNVM：通过稀疏编码和误差缓解，最大限度提高 DNN 存储密度和推理效率。发表于 MICRO。[58] Eric Qin、Ananda Samajdar、Hyoukjun Kwon、Vineet Nadella、Sudarshan Srinivasan、Dipankar Das、Bharat Kaul 和 Tushar Krishna。2020年。SIGMA：一种用于 DNN 训练、具备灵活互连的稀疏非规则 GEMM 加速器。发表于 HPCA。[59] Minsoo Rhu、Natalia Gimelshein、Jason Clemons、Arslan Zulfiqar 和 Stephen W. Keckler。2016年。vDNN：面向可扩展、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。[60] Fazle Sadi、Joe Sweeney、Tze Meng Low、James C. Hoe、Larry Pileggi 和 Franz Franchetti。2019年。利用可扩展的多路归并并行化，高效执行大型且高度稀疏矩阵的 SpMV 运算。发表于 MICRO。[61] Ananda Samajdar、Yuhao Zhu、Paul Whatmough、Matthew Mattina 和 Tushar Krishna。2018年。SCALE-Sim：脉动阵列 CNN 加速器模拟器。arXiv 预印本（2018年）。[62] Prithviraj Sen、Galileo Namata、Mustafa Bilgic、Lise Getoor、Brian Galligher 和 Tina Eliassi-Rad。2008年。网络数据中的集体分类。AI Magazine 29，3（2008年9月），93。https://doi.org/10.1609/aimag.v29i3.2157[63] Yakun Sophia Shao、Jason Clemons、Rangharajan Venkatesan、Brian Zimmer、Matthew Fojtik、Nan Jiang、Ben Keller、Alicia Klinefelter、Nathaniel Pinckney、Priyanka Raina、Stephen G. Tell、Yanqing Zhang、William J. Dally、Joel Emer、C. Thomas Gray、Brucek Khailany 和 Stephen W. Keckler。2019年。Simba：采用基于多芯片模块的架构扩展深度学习推理。发表于 MICRO。14–27。[64] Franyell Silfa、Gem Dot、Jose-Maria Arnau 和 Antonio Gonazàlez。2019年。循环神经网络中的神经元级模糊记忆化。发表于 MICRO。[65] Kevin Siu、Dylan Malone Stuart、Mostafa Mahmoud 和 Andreas Moshovos。2018年。卷积神经网络硬件加速器的内存需求。发表于 IISWC。[66] Nitish Srivastava、Hanchen Jin、Shaden Smith、Hongbo Rong、David Albonesi 和 Zhiru Zhang。2020年。Tensaurus：一种用于混合稀疏-稠密张量计算的多功能加速器。发表于 HPCA。[67] Lubos Takac 和 Michal Zabovsky。2012年。公共社交网络中的数据分析。发表于 DTI。[68] Rajeev Thakur、Rolf Rabenseifner 和 William Gropp。2005年。MPICH 中集合通信操作的优化。IJHPCA（2005年）。[69] Shyamkumar Thoziyoor、Naveen Muralimanohar、Jung Ho Ahn 和 Norman P Jouppi。2008年。CACTI 5.1。技术报告。2008-20，HP实验室。[70] Shikhar Vashishth。2019年。自然语言处理的神经图嵌入方法。arXiv 预印本（2019年）。[71] Kuansan Wang、Zhihong Shen、Chiyuan Huang、Chieh-Han Wu、Yuxiao Dong 和 Anshul Kanakia。2020年。微软学术图谱：专家不足时该怎么办。定量科学研究 1，1（2020年），396–413。[72] Yang Wang、Chen Zhang、Zhiqiang Xie、Cong Guo、Yunxin Liu 和 Jingwen Leng。2021年。双侧稀疏张量核心。发表于 ISCA。[73] Xuechao Wei、Yun Liang 和 Jason Cong。2019年。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。[74] Keyulu Xu、Weihua Hu、Jure Leskovec 和 Stefanie Jegelka。2018年。图神经网络的表达能力有多强？arXiv 预印本（2018年）。[75] Mingyu Yan、Lei Deng、Xing Hu、Ling Liang、Yujing Feng、Xiaochun Ye、Zhimin Zhang、Dongrui Fan 和 Yuan Xie。2020 年。HyGCN：采用混合架构的 GCN 加速器。载于 *HPCA*。[76] Pinar Yanardag 和 SVN Vishwanathan。2015 年。深度图核。载于 *KDD*。[77] Jaewon Yang 和 Jure Leskovec。2015 年。基于真实标签定义和评估网络社群。*Knowledge andInformation Systems* 42，1（2015），181–213。[78] Xuan Yang、MingyuGao、JingPu、AnkitaNayak、Qiaoyi Liu、Steven Emberton Bell、Jeff Ou Setter、Kaidi Cao、Heonjae Ha、Christos Kozyrakis 和 Mark Horowitz。2018 年。DNN 数据流选择被高估了。*arXiv preprint*（2018）。[79] Yifan Yang、J. Emer 和 Daniel Sánchez。2021 年。SpZip：为不规则应用提供有效数据压缩的体系结构支持。载于 *ISCA*。[80] Pengcheng Yao、Long Zheng、Zhen Zeng、Yu Huang、Chuangyi Gui、Xiaofei Liao、Hai Jin 和 Jingling Xue。2020 年。面向图挖掘应用的局部性感知节能加速器。载于 *MICRO*。[81] Muhan Zhang 和 Yixin Chen。2018 年。基于图神经网络的链接预测。载于 *NeurIPS*。[82] Mingxing Zhang、Youwei Zhuo、Chao Wang、Mingyu Gao、Yongwei Wu、Kang Chen、Christos Kozyrakis 和 Xuehai Qian。2018 年。GraphP：通过高效数据分区减少基于处理器内存的图处理中的通信。载于 *HPCA*。[83] Shijin Zhang、Zidong Du、Lei Zhang、Huiyang Lan、Shaoli Liu、Ling Li、Qi Guo、Tianshi Chen 和 Yunji Chen。2016 年。Cambricon-X：稀疏神经网络加速器。载于 *MICRO*。[84] Xi Zhang、Chongmin Li、Haixia Wang 和 Dongsheng Wang。2010年。一种使用自适应插入和再次引用预测的缓存替换策略。载于 *2010*年第*22*届计算机体系结构与高性能计算国际研讨会。 95 102 10 1109 2010 21 – 。https://doi.org/ 。/[85] SBAC-PAD。 。Zhekai Zhang、Hanrui Wang、Song Han 和 William J. Dally。2020 年。SpArch：稀疏矩阵乘法的高效架构。载于 *HPCA*。[86] Da Zheng、Disa Mhembere、Vince Lyzinski、Joshua T Vogelstein、Carey E Priebe 和 Randal Burns。2016 年。面向十亿节点图的半外存稀疏矩阵乘法。*TPDS*（2016）。[87] Xuda Zhou、Zidong Du、Qi Guo、Shaoli Liu、Chengsi Liu、Chao Wang、Xuehai Zhou、Ling Li、Tianshi Chen 和 Yunji Chen。2018 年。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。载于 *MICRO*。[88] Maohua Zhu、Tao Zhang、Zhenyu Gu 和 Yuan Xie。2019 年。稀疏张量核心：现代图形处理器上向量级稀疏神经网络的算法与硬件协同设计。载于 *MICRO*。[89] Xiaowei Zhu、Wentao Han 和 Wenguang Chen。2015 年。GridGraph：使用两级分层划分在单机上进行大规模图处理。载于 *ATC*。[90] Youwei Zhuo、Chao Wang、Mingxing Zhang、Rui Wang、Dimin Niu、Yanzhi Wang 和 Xuehai Qian。2019 年。GraphQ：可扩展的基于处理器内存的图处理。载于 *MICRO*。