HyScale-GNN：面向单节点异构架构的可扩展混合GNN训练系统

Yi-Chien Lin、Viktor Prasanna 南加州大学，美国
加利福尼亚州洛杉矶 邮箱：{yichienl, prasanna}
@usc.edu

arXiv:2303.00158v1 [cs.DC] 1 Mar 2023

摘要——**图神经网络（GNN）已成功应用于许多涉及图结构数据的实际场景。大多数现有的单节点GNN训练系统能够训练包含数千万条边的中等规模图；然而，要将其扩展至包含数十亿条边的大规模图，仍然颇具挑战。此外，将GNN训练算法映射到计算节点上也并非易事，因为最先进的机器采用异构架构，由多个处理器和多种加速器组成。**

我们提出了HyScale-GNN，这是一种用于在单节点异构架构上训练GNN模型的新型系统。HyScale-GNN采用混合训练，利用处理器和加速器协同训练模型。我们的系统设计克服了现有工作的内存容量限制，并针对大规模图上的GNN训练进行了优化。我们提出了一种两阶段数据预取方案，以减少图神经网络训练期间的通信开销。为提高任务映射效率，我们提出了一种动态资源管理机制，可在运行时调整工作负载分配和资源分配。我们在CPU-GPU和CPU-FPGA异构架构上评估了HyScale-GNN。我们采用多个大规模数据集和两个广泛使用的GNN模型，将本设计的性能与基于PyTorch-Geometric实现的多GPU基线进行比较。CPU-GPU设计和CPU-FPGA设计分别实现了最高 $2.08\times$ 加速比和 $12.6\times$ 加速比。与 P^3 和DistDGL等最先进的大规模多节点GNN训练系统相比，我们的CPU-FPGA设计在单节点上实现了最高 $5.27\times$ 加速比。

索引词—**图神经网络训练、异构架构、大规模图**

I. 引言

图神经网络（GNNs）已成为图上表示学习的最先进模型，可促进分子性质预测 [1]，[2], 社交推荐系统 [3], [4], 电子设计自动化 [5], [6], 等众多应用。这些领域通常涉及包含超过十亿条边的大规模图 [7]。扩展图神经网络训练系统以支持此类大规模图仍然具有挑战性。以往的工作 [8]–[11] 将输入图存储在设备内存中（例如 GPU全局内存、FPGA 本地双倍数据速率内存），而不是 CPU 内存中，因为通过 DDR通道从设备内存访问数据的速度远快于通过 PCIe 从 CPU 内存访问数据。该设置的缺点是设备内存容量有限（通常为 16 至 64 GB），因此无法容纳 MAG240M [7] （202 GB）等大规模图；将图存储在 CPU 内存中可以克服这一限制，但随后需要通过 PCIe 获取数据，而 PCIe 的内存带宽较低。除了内存容量限制之外，由于现代计算机的复杂架构，将 GNN 训练算法映射到目标平台上也颇具挑战。具体而言，先进节点采用异构架构设计，以满足各种应用的性能需求 [12], [13]。异构架构由多个相互连接的多核 CPU 和若干加速器组成；所连接的加速器可以是 GPU、FPGA 或专用人工智能加速器 [14]–[16]。现有大多数工作采用简单且静态的任务映射 [9], [17], [18] ，将 CPU 视为预处理器，其主要目的在于将 GNN 计算卸载到加速器上；这种直接的任务映射忽视了利用 CPU 资源进行训练的潜力。例如，在配备单个 Nvidia RTX A5000（27.8 TFLOPS）的双路 AMD EPYC 7763（7.2 TFLOPS）平台上，利用 CPU+GPU 进行训练，与纯 GPU 训练相比，理论上可获得 (7.2+27.8) / 27.8 = 1.26× 的加速比。此外，执行 GNN 训练任务的速度取决于训练算法和目标平台的性能；这使得静态任务映射效率低下。

针对这些挑战，我们提出了 HyScale-GNN，这是一种混合GNN 训练系统，可在给定的异构架构上高效训练 GNN模型。我们提出了一种通用的处理器-加速器训练协议，定义处理器和加速器应如何交互与同步，以协同训练 GNN 模型。该协议具有通用性，可适配 GPU 、现场可编程门阵列或专用人工智能加速器等多种加速器。我们提出了一种动态资源管理机制，可将图神经网络训练任务高效映射到给定的异构架构上。该机制将图神经网络训练任务分配给 CPU 和加速器，并在运行时动态调整工作负载分配。不同于以往导致 CPU 大部分时间处于闲置状态的工作，我们的混合训练系统可充分利用 CPU 和加速器协同训练 GNN 模型。此外，HyScale-GNN 支持在包含数十亿条边的大规模图上进行图神经网络训练。为支持大规模图，我们的系统将输入图存储在 CPU 内存中；在高端节点上，CPU 内存容量可达数 TB。为减少从 CPU 内存读取数据所产生的高昂 PCIe 通信开销，我们提出了一种两阶段特征预取方案，将所需数据预先加载到加速器上。尽管我们

表I 图神经网络的
符号说明

符号	说明	符号	说明
$\mathcal{G}(\mathcal{V},\ \mathcal{E})$	输入图拓扑	$\boldsymbol{h}_i^l$	v_i 在第 l 层的特征向量
$\mathcal{V}$	顶点集	$\boldsymbol{a}_i^l$	v_i 在第 l 层的聚合结果
$\mathcal{E}$	边集	L	图神经网络层数
$\boldsymbol{X}$	输入特征矩阵	f^l	第 l层的特征长度
$\mathcal{V}^l$	第 l层的采样顶点	$\mathcal{N}(i)$	v_i的邻居
$\mathcal{E}^l$	第 l层的采样边	$\phi(.)$	逐元素激活
$\boldsymbol{X}'$	采样顶点的特征矩阵	$\boldsymbol{W}^l$	第 l层的权重矩阵

在系统中采用了多种优化，但这些优化不会改变图神经网络训练算法的语义；因此，其收敛速度和模型准确率与原始顺序算法保持一致。

我们将本工作的贡献总结如下:

- 我们提出了 HyScale-GNN，这是一种混合图神经网络训练系统，可高效利用 CPU 和加速器协同进行图神经网络训练。与现有工作相比，我们的系统实现了相同的收敛速度和模型准确率 [19]–[21] ，因为所提出的优化不会改变原有训练算法。
- 我们提出了一种通用的处理器—加速器训练协议，使 HyScale-GNN 能够适配多种加速器，包括 GPU、FPGA 或专用 AI 加速器。
- 为支持在大规模图（如 ogbn-papers100 [22] 和 MAG240M [7]）上进行图神经网络训练，我们提出将输入图存储在 CPU 内存中，并执行两阶段数据预取以隐藏通信开销。
- 我们提出了一种性能模型，该模型基于 GNN 训练算法的算法参数和平台元数据，预测系统的训练性能。HyScale-GNN 利用预测的性能，在设计阶段将任务进行粗粒度映射到目标平台上。
- 我们提出了一种动态资源管理机制，在运行时通过微调分配给 CPU 和加速器的工作负载来执行细粒度任务映射。
- 我们使用多个大规模图和两种广泛使用的GNN模型: GraphSAGE[2], 和图卷积网络（GCN） [23]对 HyScale-GNN进行了评估。在连接4个高端GPU的双路平台以及连接4个高端FPGA的双路平台上，与使用 PyTorch-Geometric [18],实现的多GPU基线相比，我们的CPU-GPU和CPU-FPGA设计分别实现了最高 2.08× 的加速比和12.6× 的加速比。与使用多节点集群中16至64个GPU的最先进水平分布式GNN训练系统 [19], [21]相比，我们的CPU-FPGA设计仅使用单节点上的4个FPGA即可实现最高 5.2×的加速比。

II. 背景

A. 图神经网络

我们在表I中定义了与GNN相关的符号。GNN学习为顶点集（即目标顶点$\mathcal{V}^L$）生成低维向量表示（即节点嵌入），节点嵌入可以促进许多下游应用，如第I节所述。GNN模型可以使用聚合-更新范式 [24]表示为:

$$\boldsymbol{a}_v^l = \text{AGGREGATE}(\boldsymbol{h}_u^{l-1} : u \in \mathcal{N}(v) \cup \{v\}) \tag{1}$$

$$\boldsymbol{h}_v^l = \phi(\text{UPDATE}(\boldsymbol{a}_v^l, \boldsymbol{W}^l)) \tag{2}$$

在特征聚合阶段，对于每个顶点 v，将邻居顶点 $u \in \mathcal{N}(v)$ 的特征向量 h_u^{l-1} 使用均值、最大值或长短期记忆网络等算法特定算子聚合到 a_v^l 中。由于图结构数据是非欧几里得的，访问邻居顶点的特征向量 h_u^{l-1} 会产生大量不规则数据访问。特征更新阶段由多层感知机（MLP）和逐元素激活函数 ϕ （例如修正线性单元）组成，分别对 a_v^l 应用线性变换和非线性变换。尽管存在多种 GNN模型，但这些模型都遵循聚合-更新范式。我们列出两个具有代表性的模型作为示例:

- 图卷积网络 [23]: 是应用最广泛的图神经网络模型之一。该模型可表述如下:

$$\begin{aligned} \boldsymbol{a}_v^l &= \text{Sum}\left(\frac{1}{\sqrt{D(v)\cdot D(u)}} \cdot \boldsymbol{h}_u^{l-1}\right) \\ \boldsymbol{h}_v^l &= \text{ReLU}\left(\boldsymbol{a}_v^l \boldsymbol{W}^l + \boldsymbol{b}^l\right) \end{aligned} \tag{3}$$

其中， $D(v)$表示顶点 v的度， $\boldsymbol{b}^l$表示更新函数的偏置。

- GraphSAGE [2]: 提出了一种用于小批次图神经网络训练的邻居采样算法。该模型可表述如下:

$$\begin{aligned} \boldsymbol{a}_v^l &= \boldsymbol{h}_v^{l-1} || \text{Mean}\left(\boldsymbol{h}_u^{l-1}\right) \\ \boldsymbol{h}_v^l &= \text{ReLU}\left(\boldsymbol{a}_v^l \boldsymbol{W}^l + \boldsymbol{b}^l\right) \end{aligned} \tag{4}$$

其中， $||$ 表示拼接操作。

通过在我们的系统设计中采用聚合-更新范式，我们的工作能够训练各种图神经网络模型。

B. 小批次图神经网络训练

我们在图1中展示了小批次图神经网络训练的工作流程。在每次训练迭代中，采样器首先从原始图 $\mathcal{G}(\mathcal{V},\ \mathcal{E})$ 中提取一个小批量 $\{\mathcal{G}(\mathcal{V}^l,\ \mathcal{E}^l) :\ 1 \leqslant l \leqslant L\}$ 。该小批量作为计算图，用于执行图神经网络操作，即特征聚合和特征更新。在前向传播阶段，图神经网络操作会迭代执行 L 次。输出嵌入 $\{\boldsymbol{h}_i^L :\ v_i \in \mathcal{V}^L\}$ 与真实值进行比较，以计算损失。计算得到的损失随后用作反向传播的输入。反向传播以相反的方向 [25] 执行与前向传播相同的一组图神经网络操作；反向传播会生成

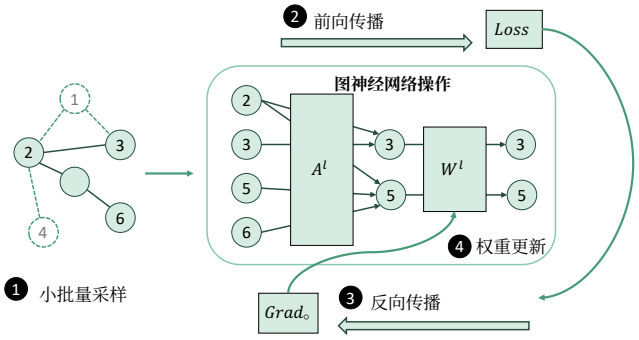

图 1. 小批次图神经网络训练概述

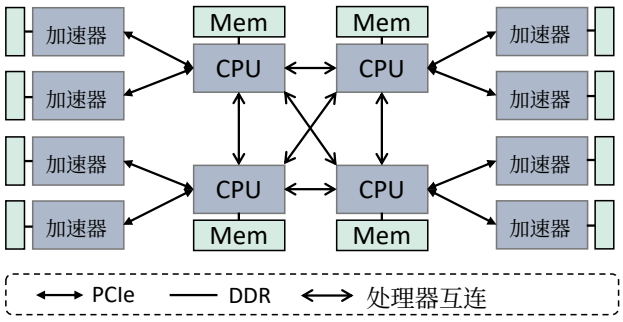

图 2. 目标异构架构

每一层中权重矩阵 W^l 的梯度，随后利用这些梯度更新模型权重。

我们的工作采用同步随机梯度下降 (SGD) [26]，在多个设备上训练图神经网络，其工作流程与原始图神经网络训练类似，但略有不同。首先，采样多个小批次，然后为每个设备分配一个小批次。接着，每个设备按照原始 GNN 训练算法执行前向/反向传播。最后，收集每个设备上的梯度并取其平均值。随后，将平均梯度广播到各个设备，以执行全局权重更新。在多个设备上使用同步随机梯度下降进行训练，在算法上等同于在单个设备上使用更大的小批量进行训练。例如，使用小批量大小为 1024 在 4 个 GPU 上训练，等同于使用小批量大小为 4096 [27] 在 1 个 GPU 上训练。

C. 目标异构架构

图 2 展示了目标异构架构。它由多个 CPU 和多个加速器组成。该平台上的 CPU 内存构成一个共享地址空间：每个 CPU 都能访问与其连接的 CPU 内存，也能通过 QPI 等处理器互连通道访问与其他 CPU 连接的 CPU 内存 [28]。每个加速器都通过 PCIe 连接到一个处理器，并通过 DDR通道连接到设备内存。

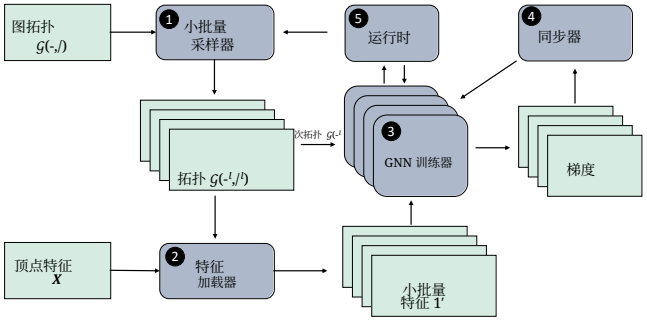

图 3. 系统概述

III. 系统设计

本节首先在第 III-A 节介绍 HyScale-GNN 的逻辑组件。接着，我们将在第 III-B 节展示这些逻辑组件如何在异构平台上运行。最后，我们将在第 III-C 节介绍处理器-加速器训练协议，该协议定义处理器和加速器应如何交互以进行混合训练。

A. 混合*GNN*训练系统

HyScale-GNN由多个逻辑组件构成。图3展示了这些逻辑组件（灰色圆角矩形）及其输入/输出（绿色矩形），下文将逐一介绍：

小批量采样器：在每次训练迭代开始时，小批量采样器以图拓扑 $\mathcal{G}(\mathcal{V},\ \mathcal{E})$为输入，并通过执行采样算法 [2], [29] 生成多个小批次。

特征加载器： 给定一个小批次，特征加载器从原始特征矩阵 $\boldsymbol{X}$中提取小批量特征矩阵 $\boldsymbol{X}'$ 。提取出的特征矩阵 $\boldsymbol{X}'$ 仅包含采样顶点的顶点特征，而非输入图中的所有顶点的特征。

GNN训练器： GNN训练器负责执行图神经网络训练的前向传播和反向传播。它们以小批量拓扑和小批量特征矩阵为输入，并生成用于模型权重更新的梯度。

同步器：每个 GNN 训练器生成一组梯度后，同步器会执行全归约操作，收集各个训练器的梯度，计算梯度的平均值，并将平均梯度广播回各个训练器，以更新其模型权重。

运行时：运行时系统根据我们的处理器-加速器训练协议（第 III-C 节），管理 CPU 与加速器之间的交互和数据通信。此外，它还执行动态资源管理，在训练期间微调目标平台上的工作负载分配。

B. 任务映射与协调

我们的混合训练系统由一个运行时线程、若干处理器线程和若干加速器线程组成。

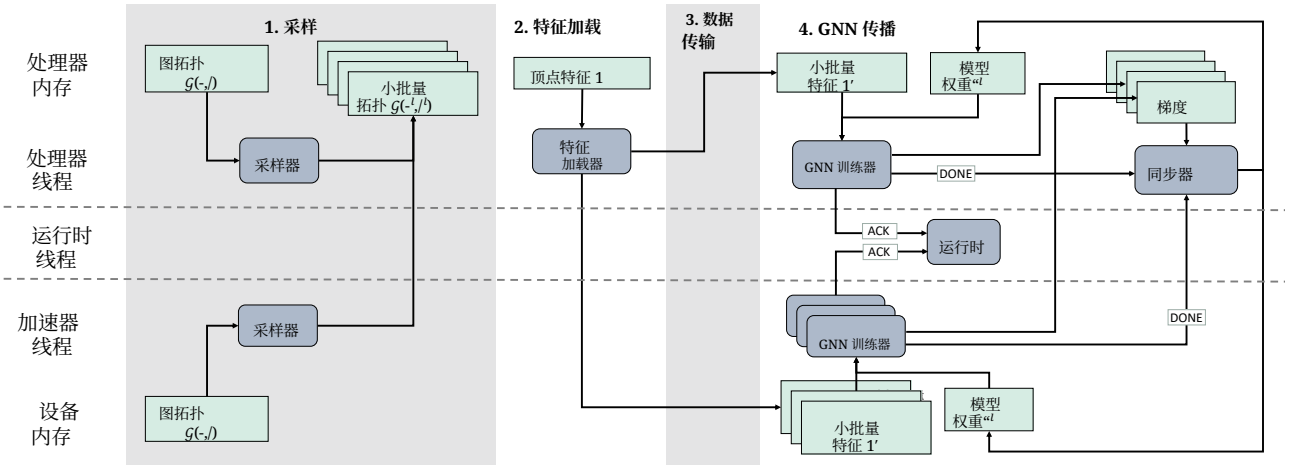

图 4. 任务映射与协调

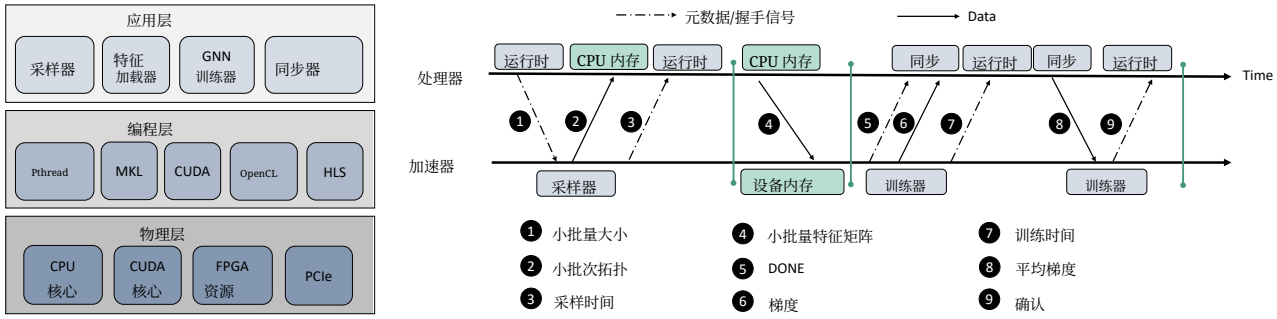

图 5. 处理器-加速器训练协议

每个逻辑组件（第 III-A 节）都映射到一个或多个线程。图 4 展示了 HyScale-GNN 的任务映射与协调。HyScale-GNN 将图神经网络训练分解为四个流水线阶段：采样、特征加载、数据传输和 GNN 传播：**(1) 采样**：小批量采样可在 CPU 和加速器上执行。每次训练迭代中会生成 n 个小批次，其中 n 是系统中的 GNN 训练器数量。每生成一个小批次，便将其存储在 CPU 内存中，以供特征加载。**(2) 特征加载**：收集完所有 n 个小批次后，特征加载器会从输入特征$\boldsymbol{X}$中读取采样顶点的特征向量，并将加载的特征存储在子矩阵 $\boldsymbol{X}'$ 中。特征加载仅在 CPU 上执行。这是因为对于大规模图，输入特征矩阵 $\boldsymbol{X}$ 过大，无法装入设备内存；因此，特征矩阵 $\boldsymbol{X}$ 存储在 CPU 内存中，由运行于 CPU 上的特征加载器访问。**(3) 数据传输**：小批次可以在 CPU 或加速器上执行。如果小批次在加速器上执行，则小批次拓扑 $\{\mathcal{G}(\mathcal{V}^l, \mathcal{E}^l) : 1 \leqslant l \leqslant L\}$ 和小批量特征矩阵 $\boldsymbol{X}'$ 会通过 PCIe 传输至加速器设备内存。**(4) GNN 传播**：在每次训练迭代中，每个设备（处理器或加速器）都会被分配一个小批次拓扑和一个小批量特征矩阵；它们充当作为 GNN 训练器执行前向传播与反向传播的输入。最初，工作负载（即小批量大小）的分配在设计时根据我们的性能模型（第 V 节）确定。如果运行时设备之间存在工作负载不均衡，DRM引擎（第 IV-A 节）会调整下一次训练迭代的工作负载分配。传播完成后，每个训练器都会生成一组梯度，随后用于更新模型权重；当梯度存储或传输到 CPU 内存后，每个训练器会向同步器发送“完成”信号。由于所有加速器都连接到 CPU，且各 CPU 彼此连接（图 2），因此在 CPU 上运行同步器是很自然的。同步器收到所有训练器发送的“完成”信号后，会执行全归约操作，对收集到的梯度求平均，并将结果广播回每个训练器。所有训练器更新其本地模型权重并向运行时系统发送确认后，运行时系统才会进入下一次训练迭代。

C. 处理器-加速器训练协议

为了在给定的异构架构上执行混合训练，我们提出了一种通用的处理器-加速器训练协议。该协议由三层组成：应用层由第

III-A 节中定义的逻辑组件构成；编程层由用于在多核 CPU、GPU、FPGA 或专用人工智能加速器上实现这些逻辑组件的库构成；物理层由实际硬件构成。由于进程与加速器的交互是在应用层定义的，不受特定加速器类型的限制，因此 HyScale-GNN 可以移植到各种异构架构上。我们在图 5 中展示了数据交换和握手信号。请注意，图 5 未描绘特征加载阶段，因为该阶段没有数据交换或握手信号。在每个流水线阶段的末尾都设置了一个屏障以进行同步。此外，运行时系统会收集各阶段的执行时间，以便在下一次迭代中微调工作负载分配（第 IV-A 节）。

IV. 优化

为了实现较高的图神经网络训练吞吐量，我们开发了多种优化方法，以实现高效的任务映射（第 IV-A 节）并降低通信开销（第 IV-B、IV-C 节）。值得注意的是，这些优化不会改变原始图神经网络训练算法的语义。因此，HyScale-GNN 不会以牺牲模型准确率和收敛速度为代价来提高训练吞吐量。

A. 动态资源管理

为了将图神经网络训练任务高效地映射到异构架构上，我们首先利用性能模型（第 V 节）给出的预测结果，在编译时初始化图神经网络训练任务映射。此外，我们提出了一个动态资源管理（*DRM*）引擎，用于微调资源分配和任务映射，以在运行时提高图神经网络训练吞吐量。DRM 引擎是一种瓶颈引导优化器，通过加速每次迭代中的瓶颈阶段来提高训练吞吐量。DRM 引擎提供两种功能来加速瓶颈阶段：工作负载均衡和线程均衡。工作负载均衡功能通过调整分配给训练器的小批量大小，来平衡 CPU 与加速器之间的工作负载。重新分配后，在混合系统上执行的小批量总大小保持不变。线程均衡功能用于探索 CPU 任务（例如 CPU 采样器、 CPU 训练器）之间的性能权衡，并且仅在瓶颈阶段为 CPU 任务时使用。它通过减少分配给最快 CPU 任务的线程数，并将这些线程重新分配给瓶颈阶段，从而加速瓶颈阶段。

算法 1 从高层次描述了 DRM引擎的工作方式。首先，DRM引擎将数据传输时间与加速器训练时间合并处理，因为二者的执行时间高度相关。例如，如果分配给加速器的工作负载减少，数据传输时间也会缩短，因为需要通信的数据更少。DRM引擎以各阶段的执行时间为输入，并识别瓶颈阶段和最快阶段。如果系统的瓶颈是加速器任务，那么 DRM 会执行 工作负载均衡 以调整

算法 1 动态资源管理 输入：加速器采样的执行时间 T_{SA}、CPU 采样 T_{SC}、特征加载 T_{Load}、数据传输 T_{Tran}、CPU 训练 T_{TC}、加速器训练 T_{TA}输出：线程分配、工作负载分配 1： $T_{Accel} = \max(T_{Tran}, T_{TA})$ **2：对所有项排序 =** $\text{sort}(T_{SC}, T_{SA}, T_{Load}, T_{TC}, T_{Accel})$ **3：最快项 =排序列表**[4] **4：第二快项 =排序列表**[3]**5：瓶颈 =排序列表**[0]**6： 7：对 CPU 项排序 =** $\text{sort}(T_{SC}, T_{Load}, T_{TC})$ **8：最快的 CPU 任务 =已排序的 CPU 项**[2]**9： 10：根据瓶颈执行 switch ▷ 瓶颈引导优化器 11：case** T_{SA}**: 12：balance work(**T_{SC}, T_{SA}**) 13：case** T_{Accel}**: 14：balance work(**T_{TC}, T_{Accel}**) 15：case** T_{Load}**: 16：balance thread(fastest CPU task, bottleneck) 17：case** T_{SC}**: 18：if fastest ==** T_{SA} **then 19：balance work(**T_{SC}, T_{SA}**) 20：else if (fastest ==** T_{Accel} **and second ==** T_{SA}**) then 21：balance work(**T_{SC}, T_{SA}**) 22：else 23：balance thread(fastest, bottleneck) 24：end if 25：case** T_{TC}**: 26：if fastest ==** T_{Accel} **then 27：balance work(**T_{TC}, T_{Accel}**) 28：else if (fastest ==** T_{SA} **and second ==** T_{Accel}**) then 29：balance work(**T_{TC}, T_{Accel}**) 30：else 31：balance thread(fastest, bottleneck) 32：end if**

下一次迭代中 CPU 与加速器之间的工作负载分配。如果系统受特征加载器限制，DRM 引擎会执行线程均衡，重新分配更多线程来执行特征加载。如果系统受 CPU 采样器限制，DRM 引擎可以执行工作负载均衡 或线程均衡，以提升 CPU 采样器的速度。具体决策取决于哪个阶段运行得最快。如果加速器采样器运行得最快，DRM 引擎会执行工作负载均衡，增加分配给加速器的工作负载；如果最快的任务是加速器训练器，其次是加速器采样器，DRM 引擎也会向加速器分配更多工作负载；否则，DRM 引擎会执行线程均衡，以探索其他 CPU 任务之间的性能权衡。最后，如果系统受 CPU 训练器限制，DRM 引擎也可以提升性能

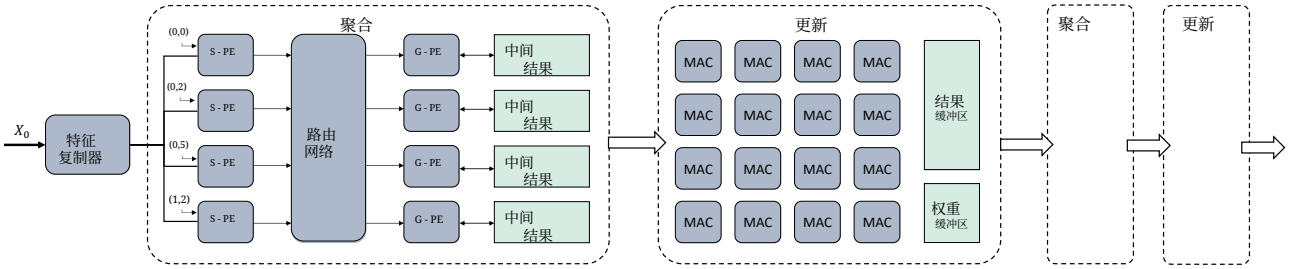

图 6. 硬件内核设计与数据通路

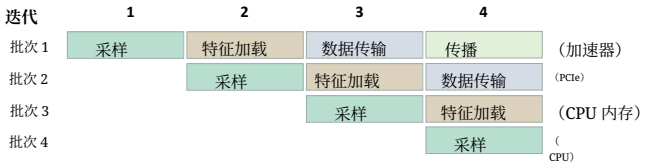

图 7. 两阶段特征预取

通过执行工作负载均衡 或线程均衡来提升性能；因此，这种改进策略与系统受 CPU 采样器限制时类似。

B. 两阶段特征预取

HyScale-GNN 在 CPU 和加速器上运行图神经网络训练。对于在加速器上运行的图神经网络训练器，数据需要先从 CPU 内存中读取，再通过 PCIe 传输到加速器。为减少高昂的通信开销，HyScale-GNN 执行特征预取。其思路是先将下一次训练迭代所需的小批次预加载到加速器上，使加速器执行图神经网络操作时无需等待数据。我们发现特征预取阶段仍可能成为系统瓶颈，因此进一步将特征预取分解为两个流水线阶段：特征加载和数据传输。特征加载阶段从 CPU 内存加载小批次特征 $\boldsymbol{X}'$，数据传输阶段则通过 PCIe 将矩阵 $\boldsymbol{X}'$ 发送到加速器。这两个阶段可以并发运行，因为它们使用不同的内存通道（CPU RAM通道和 PCIe通道），且小批次之间不存在数据依赖。图7展示了两阶段特征预取的一个示例：在第4次迭代中，加速器执行小批次1的同时，小批次2的特征矩阵正通过 PCIe 传输到加速器，小批次3的特征矩阵也在同时从 CPU 内存中加载。因此，在下一次迭代中加速器执行小批次2时，小批次拓扑和小批次特征已就绪，并存储在加速器的设备内存中。请注意，图7仅展示了系统流水线的简化版本。每次迭代都可以采样、加载、传输和执行多个小批次。还值得注意的是，我们的系统流水线能够高效利用异构架构上的各种资源。

C. 硬件内核设计

图神经网络训练会产生大量内存访问，而高昂的内存访问开销限制了训练吞吐量。因此，如图6所示，我们设计了专用硬件内核和数据通路，以减少 GNN 传播阶段的外部内存访问。GNN 传播由聚合阶段和更新阶段组成（第II-A节）。对于更新阶段，我们采用脉动阵列设计来执行多层感知机（MLP）；对于聚合阶段，我们采用散射-聚集设计 [17], [30]，以并行处理多条边。图6展示了一个内核示例，其中包含四组散射-聚集处理单元（PE），可并行处理四条边。为最大限度地复用数据，HyScale-GNN 首先按源顶点对一个小批次内的边进行排序，使源顶点相同的边能够连续执行。当从外部内存读取顶点特征时，特征复制器会将读取的特征广播到所有散射处理单元（S-PE）。随后，该特征存储在 S-PE 的本地内存中，并被复用 $D_{out}(v)$ 次，其中 $D_{out}(v)$ 是顶点 v 的出度。例如，在图6中处理四条边。假设 $D_{out}(v_0)$ 为 3，那么加载的特征 X_0最多可以复用三次。由于边按源顶点排序，前三条边的源顶点相同，为 v_0，因此 X_0 会被复用三次。第四个 S-PE 将保持空闲，直到从内存中读取 X_1。这种设计最大限度地复用了输入数据，因为每个顶点特征只需从内存读取一次，并将输入内存流量从 $O(\mathcal{E}^1)$ 降低至 $O(\mathcal{V}^0)$（符号定义见表I）。为减少内存占用，我们设计了定制数据通路，避免在 GNN 传播期间将中间结果写回内存。如图6所示，聚合内核的输出直接发送到更新内核，而更新内核的输出则发送到聚合内核，用于下一层的特征聚合。因此，只有最终输出会写回内存。

V. 性能模型

为了在异构架构上初始化任务映射，我们提出了一种性能模型，用于预测 HyScale-GNN 的性能。首先，我们定义图神经网络训练

吞吐量为每秒遍历的边数（百万条，MTEPS）：

$$\text{Throughput} = \frac{\sum_{i=0}^{n}\sum_{l=1}^{L}|\mathcal{E}_i^l|}{T_{\text{execution}}} \tag{5}$$

我们用 n 表示系统上运行的 GNN 训练器数量。每个训练器在每次迭代中执行一个小批量。因此，分子表示所有训练器在一次迭代中遍历的边总数，分母表示一次训练迭代的执行时间（第 II-B 节）。我们的系统包含四个流水线阶段（第 III-A 节）：采样、特征加载、数据传输和 GNN 传播；因此，$T_{\text{execution}}$ 建模如下：

$$T_{\text{execution}} = \max(T_{\text{samp}}, T_{\text{load}}, T_{\text{trans}}, T_{\text{prop}}) \tag{6}$$

我们没有构建方程，而是在设计阶段通过使用不同数量的线程和不同的小批量大小运行采样算法，并推导其执行时间，以此估算 T_{samp}。这是因为不同采样算法的计算模式各不相同[2], [29], ，因此无法用一个简单的方程来对各种算法的采样时间T_{samp} 进行建模。

T_{load} 和 T_{tran} 可建模如下：

$$T_{\text{load}} = \frac{\sum_{i=0}^{n}|\mathcal{V}^0| \times f^0 \times S_{\text{feat}}}{BW_{\text{DDR}}} \tag{7}$$

$$T_{\text{trans}} = \frac{|\mathcal{V}^0| \times f^0 \times S_{\text{feat}}}{BW_{\text{PCIe}}} \tag{8}$$

公式 7 中的分子表示需要从 CPU 内存加载的顶点特征大小，而公式 8 中的分子表示需要传输到单个加速器的顶点特征大小。S_{feat} 表示数据大小，单精度浮点数占 4 字节。对于 T_{load}，数据从 CPU 内存加载，因此分母为 CPU 内存带宽；对于 T_{tran}，分母为 PCIe 带宽。我们用“带宽”表示执行突发数据传输时的有效带宽，而非峰值带宽。

多个 GNN 训练器并行运行，其 T_{train} 可建模为：

$$T_{\text{prop}} = \max_{i \in n}(T_{\text{Trainer},i}) + T_{\text{sync}} \tag{9}$$

单个训练器的执行时间可建模为：

$$\begin{aligned} T_{\text{Trainer}} = t_{\text{forward_prop}} + t_{\text{backward_prop}} = \\ \sum_{l=1}^{L} \oplus(t_{\text{aggregate}}^l, t_{\text{update}}^l) + \\ t_{\text{update}}^1 + \sum_{l=2}^{L} \oplus(t_{\text{aggregate}}^l, t_{\text{update}}^l) \end{aligned} \tag{10}$$

即执行前向传播和反向传播的总时间。$\oplus$ 运算符取决于内核设计。如果特征聚合和特征更新采用流水线处理（例如，运行于现场可编程门阵列上的训练器），那么 $\oplus$ 是 max 运算符；如果它们不采用流水线处理（例如，运行于 CPU 上的训练器），那么 $\oplus$ 是 $\sum$运算符。$t_{\text{aggregate}}^l$ 和 t_{update}^l 可以建模为：

$$t_{\text{aggregate}}^l = \frac{|\mathcal{E}^{l-1}| \times f^l \times S_{\text{feat}}}{BW_{\text{DDR}}} \tag{11}$$

$$t_{\text{update}}^l = \frac{|\mathcal{V}^l| \times f^l \times f^{l+1}}{N \times freq.} \tag{12}$$

聚合时间 $t_{\text{aggregate}}^l$ 建模为获取源顶点特征向量所需的数据流量除以内存带宽。内存带宽取决于训练器所在的位置：对于 CPU 训练器，它从 CPU 内存获取数据；对于加速器训练器，它从设备内存获取数据。由于在第 l 层特征聚合期间会处理 $|\mathcal{E}^l|$ 条边，因此数据流量可建模为 $|\mathcal{E}^{l-1}| \times f^l \times S_{\text{feat}}$。更新时间$t_{\text{update}}^l$ 建模为特征更新期间执行的乘加运算（MAC）次数除以 GNN 训练器的计算能力。我们将计算能力建模为 $N \times freq.$，其中 N 是每个训练器中的计算并行度（例如乘加单元），而$freq.$ 是工作频率。T_{sync} 可建模为：

$$T_{\text{sync}} = \frac{\sum_{l=1}^{L} f^{l-1} \times f^l \times S_{\text{feat}}}{BW_{\text{PCIe}}} \times 2 \tag{13}$$

分子是模型大小（即所有权重矩阵的总大小）。系数 2 来自 all-reduce 操作：模型首先被收集并求平均，然后再分散回每个训练器；因此，数据会通过 PCIe 传输两次。分母是 PCIe 带宽。

VI. 实验结果

A. 实验设置

1）环境：我们的实验在一台双路服务器上进行，该服务器配备两颗 AMD EPYC 7763 CPU。我们使用两种异构配置评估 HyScale-GNN：CPU-GPU异构架构和 CPU-FPGA异构架构。在 CPU-GPU异构架构中，双路服务器连接四块 Nvidia A5000 GPU；在 CPU-FPGA异构架构中，双路服务器连接四块赛灵思 U250 FPGA。表 II 列出了 CPU、GPU 和 FPGA 的详细规格。我们使用 Python v3.8 、 PyTorch v1.11、CUDA v11.3 和 PyTorch-Geometric v2.0.3 实现多 GPU基线和 CPU-GPU 设计。我们使用 Xilinx Vitis HLS v2021.2 [31]开发 FPGA内核。

表II 平台规格

平台	CPU AMD EPYC 7763	GPU Nvidia RTX A5000	FPGA Xilinx Alveo U250
技术	台积电 7 纳米+	三星 8纳米	台积电 16纳米
频率	2.45 GHz	2000 MHz	300 MHz
峰值性能	3.6 TFLOPS	27.8 TFLOPS	0.6 TFLOPS
片上存储器	256 MB 三级缓存	6 MB 二级缓存	54 MB
内存带宽	205 GB/s	768 GB/s	77 GB/s

表 III 数据集和 GNN 层维度统计

数据集	#顶点	#边	f_0	f_1	f_2
ogbn-products	2,449,029	61,859,140	100	256	47
ogbn-papers100M	111,059,956	1,615,685,872	128	256	172
MAG240M（同质）	121,751,666	1,297,748,926	756	256	153

表IV 硬件参数与资源利用率

并行度（n,m）	LUTs	DSPs	URAM	BRAM
（8, 2048）	72%	90%	48%	40%

2）*GNN*模型和数据集：我们选择两种广泛使用的图神经网络模型：图卷积网络 [23], 和 GraphSAGE [2]，以评估我们的系统。我们采用常见的模型配置：具有两层、隐藏特征维度为 256 的模型。我们选择一个中等规模数据集，以及两个边数超过十亿的大规模数据集进行评估：ogbn-products、ogbn-papers100M [22],和 MAG-240M（同质） [7]。ogbn-products 数据集是一个包含 6000 万条边的中等规模图；我们纳入该数据集，以便将我们的性能与先前的研究进行比较。MAG-240M（同质）是 MAG-240M 数据集的同质版本，仅保留原始异质图中的一种顶点类型和一种边类型。需要注意的是，MAG-240M（同质）仍包含 13 亿条边，因此属于大规模图。数据集和图神经网络层维度的详细信息见表 III。我们使用邻居采样器 [2] 生成小批次；小批量大小设为 1024，每层的邻居采样数量分别为 25 和 10。

B. 系统实现

我们在列表1中展示了如何使用编程层（第III-C节）中的库来实现处理器-加速器训练协议；虽然我们以GPU和现场可编程门阵列为例，但如果将该协议适配到其他人工智能加速器，处理器与加速器之间的交互方式也是类似的。我们使用Pthreads实现运行时系统。我们启动多个线程来交换数据、执行握手或启动加速器内核。数据传输和内核启动可以通过编程库提供的应用程序接口实现，例如CUDA和OpenCL。为实现握手，我们利用Pthreads中的条件等待函数。例如，同步器需要等待所有训练器完成GNN传播后，才能对梯度进行平均。每个训练器在生成梯度后都会递增“完成”变量，然后提示同步器。当“完成”等于训练器数量时，同步器便对收集到的梯度进行平均。

我们在表 IV 中列出了 FPGA 设计的硬件参数与资源利用率。我们用 n 和 m 分别表示聚合内核和更新内核的并行度。具体而言， n 表示.....的数量

```
1 //将数据发送到加速器 2 q.enqueueMigrateMemObjects(input, 0) //
FI
cu
q.
Ol
si
q.
di
st
ge
st
m
w
pt
CI
成
m
```

清单 1. 系统实现

散射-聚集处理单元。 m 表示基于脉动阵列的内核设计中的乘加单元数量。图6展示了 $n = 4$ 和 $m = 16$的一个示例。

C. 性能模型评估

我们通过比较预测的轮次时间与实际实验结果来评估性能模型。图8展示了在不同 FPGA 数量下，MAG240M（同质）数据集的轮次时间对比。平均预测误差介于 5% 至 14% 之间。误差来自模型中未纳入的额外延迟。首先，在加速器上启动内核时会产生初始开销。其次，模型未包含流水线刷新 [32] 的开销。这两项开销难以预测，因为它们取决于多种因素，例如目标加速器和编译器版本。

D. 可扩展性

我们使用性能模型（第 V 节）评估 HyScale-GNN 的可扩展性。图9展示了 HyScale-GNN 的可扩展性。以 CPU-FPGA平台为例，HyScale-GNN 展现出良好的可扩展性，可扩展至

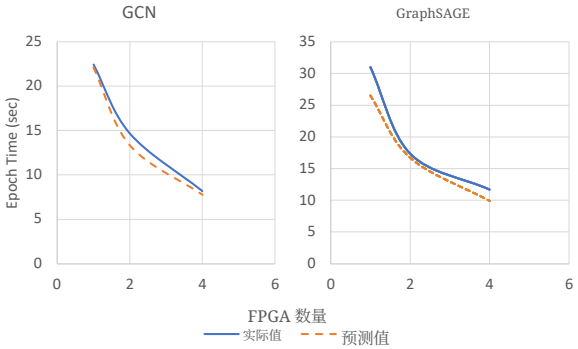

图 8. 两种 GNN模型的预测性能与实际值性能对比

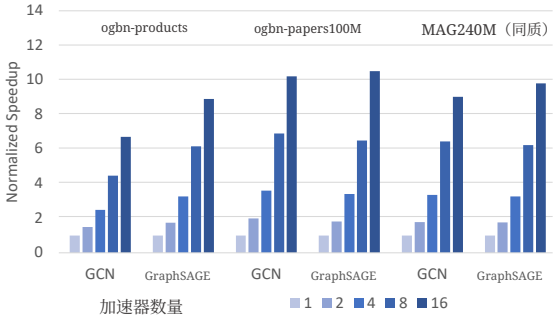

图 9. 我们的混合训练系统的可扩展性

16 个 FPGA。限制可扩展性的因素是 CPU 内存带宽。随着加速器数量增加，需要从 CPU 内存加载更多小批量特征矩阵。我们观察到，在异构平台上使用超过 12 个加速器时，CPU 内存开始达到饱和。第一组实验在 ogbn-products 数据集上运行图卷积网络模型，其可扩展性低于其他实验组。这是因为第一组实验受数据传输时间（即 PCIe 带宽）限制，从而限制了可分配给加速器的工作负载量，进而限制了可实现的加速比。

E. 整体性能

1) 性能评估：我们在 CPU-GPU异构架构和 CPU-FPGA异构架构上评估 HyScale-GNN 的性能。我们将 HyScale-GNN 的轮次时间与一种基于 PyTorch Geometric（PyG） [18] 的先进多GPU图神经网络训练实现进行比较。PyG 基线同样运行在 CPU-GPU异构架构上；不过，它并未利用 CPU 进行混合训练，因此我们将其视为多GPU基线。结果如图10所示。通过应用各种优化并进行 CPU-GPU 混合训练，与多GPU基线相比，HyScale-GNN 的加速比最高可达 $2.08\times$ 。我们将在第 VI-F节讨论各项优化的有效性。在 CPU-FPGA异构架构上，与多GPU基线相比，HyScale-GNN 的加速比最高可达 $12.6\times$ ，并且

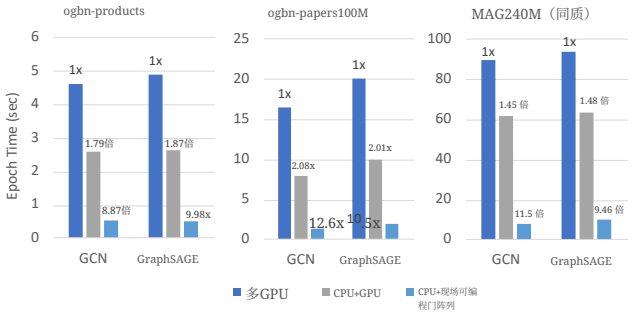

图 10. 跨平台对比

表 V 最先进水平的平台配置

	数量 计算节点	每个节点的配置	样本规模	隐藏层 dim.
PaGraph [20]	1	2 Xeon Platinum 8163 8 Nvidia V100	(25, 10)	256
P^3 [19]	4	1 至强 E5-2690 4 英伟达 P100 (2016)	(25, 10)	32
DistDGLv2 [35]	8	96 个虚拟 CPU 8 张英伟达 T4	(15, 10, 5)	256
本工作	1	2 个 EPYC 7763 4 个赛灵思 U250	-	-

$5\times-6\times$的加速比，相较于 CPU-GPU异构架构。这是因为 FPGA 具有可定制的数据通路和内存组织，使加速器训练器能够在图神经网络训练期间尽可能减少外部内存访问。具体而言，所有中间结果都利用 FPGA 丰富的片上存储器存储在芯片上，只有最终结果才会写回内存。相比之下，GPU 在整个训练过程中都需要频繁进行内存访问，因为传统缓存策略无法捕捉图神经网络训练中的数据访问模式 [33]。

2）与最先进水平的比较：已有许多工作 [9]、[17]–[21], [34], [35] 被提出用于加速图神经网络训练。然而，其中只有少数工作能够在大规模图上训练 GNN模型。我们选择了三个具有代表性的图神经网络训练系统进行比较，即 PaGraph [20]、 P^3 [19] 和 DistDGLv2 [35]。表 V 列出了各项工作的 平台配置。我们使用 *SAGE* 表示 GraphSAGE [2] 模型。在这三个大规模图神经网络训练系统中，PaGraph 是唯一运行在单节点上的系统； P^3 而 DistDGLv2 则分别运行在包含四个节点和八个节点的分布式平台上。

我们将仅使用 4 个 FPGA 在单节点上运行的系统的轮次时间，与上述训练系统进行比较。对于每组实验，我们均采用与对比系统相同的模型配置（样本规模、隐藏维度）。如表 VI 所示，与 PaGraph 和 P^3 相比，HyScale-GNN 的加速比最高分别达到 $6.9\times$ 和 $5.27\times$ 。为确保比较公平，我们根据平台峰值性能对轮次时间进行归一化；这一指标体现的是系统设计本身的有效性和效率，而不是依赖强大的硬件来

表 VI 每轮时间（秒）与最先进水平的比较

	ogbn-products		ogbn-papers100M		几何平均值
	GCN	SAGE	GCN	SAGE	加速比
PaGraph	1.18	0.25	4.00	1.18	1x
本工作	0.27	0.49	0.58	1.91	1.76x
P^3	1.11	1.23	2.61	3.11	1x
本工作	0.27	0.28	0.57	0.59	4.57x
DistDGL v2	-	0.30	-	4.16	1x
本工作	-	1.69	-	3.67	0.45倍

表 VII 与最先进水平的归一化轮次时间（秒×TFLOPS）对比

	ogbn-products		ogbn-papers100M		几何平均值
	GCN	SAGE	GCN	SAGE	加速比
PaGraph	135.1	28.63	458.0	135.1	1x
本工作	2.59	4.70	5.55	18.34	21x
P^3	165.1	183.0	388.4	462.8	1x
本工作	2.59	2.69	5.47	5.66	71x
DistDGL v2	-	163.2	-	2263	1x
本工作	-	16.20	-	35.23	25x

实现高性能。如表 VII 所示，归一化后，HyScale-GNN 相较于最先进的系统实现了 21×-71× 的加速比。HyScale-GNN 能够实现加速，原因有以下几点：（1）资源利用率：HyScale-GNN 同时利用处理器和加速器协同训练图神经网络模型。具体而言，HyScale-GNN 同时利用 CPU核心和加速器进行计算，并同时利用 CPU 内存和设备内存读取数据。我们的 DRM 引擎（第 IV-A 节）进一步确保任务能够高效映射到平台上。另一方面，PaGraph 和 P^3 没有利用平台上的处理器。（2）通信开销：如第 VI-E1 节所述，与基于 GPU 的解决方案相比，基于 FPGA 的解决方案能够有效降低外部内存访问开销。此外，PaGraph 仅将部分顶点特征缓存于设备内存中，遇到缓存未命中时需要从 CPU 内存获取数据；因此，在 ogbn-papers100M 等大规模图上训练时，由于缓存未命中频繁，PCIe 通信开销会显著增加。 P^3 会产生节点间数据通信，因为图被划分并分布到各个节点上；与 HyScale-GNN 相比，这会带来额外的通信开销。与在八个节点上运行、总计配备 64 个 GPU 的 DistDGLv2 相比，HyScale-GNN 仅使用单节点机器上的 4 个 FPGA，就能达到其性能的 $0.45\times$ 。DistDGLv2 同时利用处理器和加速器协同训练图神经网络模型。然而，DistDGLv2 采用静态任务映射，可能导致效率低下。此外，DistDGLv2 会划分输入图，并将各分区分发到每个节点，因此会像 P^3 一样产生节点间通信开销。

F. 消融实验

在本节中，我们评估应用于 HyScale-GNN 的优化措施的有效性。图11展示了在 CPU-FPGA异构架构上的评估结果；在 CPU-GPU异构架构上的评估也显示出类似结果。我们从一个基线设计开始，该设计
采用传统的任务映射方式，将大部分任务（采样、同步等任务除外）卸载到现场可编程门阵列上。随后，我们采用静态任务映射进行 CPU-FPGA 混合训练，从而实现最高 $1.13\times$ 加速比。应用 DRM

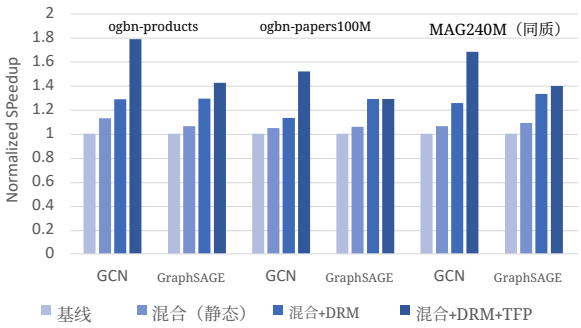

图 11. 优化的影响

优化（第 IV-A 节）后，系统的加速比最高可达 。应用 TFP（第 IV-B 节）优化后，HyScale-GNN 的加速比最高可达 $1.79\times$。这是因为数据加载阶段通常是图神经网络训练的瓶颈；如果训练主要受图神经网络传播阶段的影响（例如图 11中 ogbn-papers100M 上的 GraphSAGE 模型），那么 TFP 优化就无法带来加速。

VII. 相关工作

已有多项工作被提出 [9], [17], [34] 以加速单节点上的图神经网络训练。然而，这些工作专注于使用单个加速器执行图神经网络训练，不支持使用多个加速器进行训练。此外，GraphACT [9]和 HP-GNN [17] 等工作将输入图存储在设备内存中，因此无法支持 [7] 超出设备内存容量的大规模图。最近，已有多项工作 [19], [21], [36] 被提出，以在多节点平台上训练图神经网络。然而，这些工作需要进行图划分，从而带来工作负载不均衡和较高的节点间通信开销等问题。此外，图划分可能会影响收敛速度和模型准确率 [21]。在本工作中，我们展示了在单节点上训练大规模图神经网络并实现高训练吞吐量的可行性。

VIII. 结论

在本工作中，我们提出了 HyScale-GNN，这是一种针对大规模图上的 GNN 模型训练进行优化的混合训练系统。我们提出了多种优化方法，以降低

通信开销并实现高效的任务映射。与多GPU基线相比，我们的系统实现了最高 $12.6\times$ 的加速比。此外，仅使用单节点上的四个 FPGA，HyScale-GNN 相较于采用 8 至 16 个 GPU 的最先进训练系统，可实现 $1.76\times - 4.57\times$ 的加速比。

我们还发现 HyScale-GNN 存在一些局限性。首先，如果性能瓶颈出现在数据传输阶段（即受 PCIe 带宽限制），HyScale-GNN 就无法提供有效的解决方案。在这种情况下，DRM引擎会减少分配给加速器的工作负载，从而限制系统能够达到的加速比和可扩展性。其次，HyScale-GNN 无法直接扩展到包含多个节点的分布式平台。如果像 DistDGL 那样将输入图划分并分发到各个节点，就需要进行节点间通信和同步。然而，我们的协议定义了单个节点上处理器与加速器之间的交互方式，并不支持节点间通信。未来，我们计划采用数据量化等技术来缓解 PCIe 带宽压力，并定义一种更通用的协议，以便在分布式异构架构上训练 GNN模型。

致谢

本工作获得了美国国家科学基金会（NSF）通过 SaTC-2104264 和 OAC-2209563 号资助，以及 DEVCOM 陆军研究实验室（ARL）通过 W911NF2220159 号资助。

参考文献

[1] C.-I. Yang 和 Y.-P. Li，“深度学习分子性质预测的可解释不确定性量化”，*Journal of Cheminfor- matics*，2023。[2] W. L. Hamilton、R. Ying 和 J. Leskovec，“大规模图上的归纳式表示学习”，载于神经信息处理系统第*31*届国际会议论文集，2017。[3] R. Ying、R. He、K. Chen、P. Eksombatchai、W. L. Hamilton 和 J. Leskovec，“用于网络规模推荐系统的图卷积神经网络”，载于第*24*届*ACM*知识发现与数据挖掘国际会议论文集，2018。[4] R. Zhu、K. Zhao、H. Yang、W. Lin、C. Zhou、B. Ai、Y. Li 和 J. Zhou，“Aligraph：一个全面的图神经网络平台”，《*VLDB*会刊》，2019。[5] D. S. Lopera、L. Servadei、G. N. Kiprit、S. Hazra、R. Wille 和 W. Ecker，“面向电子设计自动化的图神经网络综述”，载于*CAD*机器学习研讨会（*MLCAD*）。IEEE，2021。[6] N. Wu、H. Yang、Y. Xie、P. Li 和 C. Hao，“使用图神经网络进行高级综合性能预测：基准测试、建模与推进”，载于设计自动化会议论文集，2022。[7] W. Hu、M. Fey、H. Ren、M. Nakata、Y. Dong 和 J. Leskovec，“Ogb-lsc：图机器学习的大规模挑战”，*arXiv*预印本 *arXiv:2103.09430*，2021。[8] Y.-C. Lin、B. Zhang 和 V. Prasanna，“在 CPU+多FPGA异构平台上加速图神经网络训练”，载于高性能计算。Springer国际出版公司，2022。[9] H. Zeng 和 V. Prasanna，“Graphact：在 CPU-FPGA异构平台上加速图卷积网络训练”，载于*2020*年*ACM/SIGDA*国际现场可编程门阵列研讨会论文集，2020。[10] C. Zhang 等，“H-gcn：基于 Versal ACAP架构的图卷积网络加速器。”国际现场可编程逻辑与应用会议，2022。[11] Y. C. Lin、B. Zhang 和 V. Prasanna，“使用高级综合加速图卷积网络推理”，载于*2021*年*IEEE*高性能极限计算会议（*HPEC*），2021。[12] “Amazon ec2 f1 [在线],” https://aws.amazon.com/tw/ec2/instance-types/f1/，访问日期：2022-06-23。[13] “Azure np-series [在线],” htt ps://docs.microsoft.com/en-us/azure/virtual-machines/np-series，访问日期：2022-06-23。[14] D. Abts 等，“面向大规模机器学习的软件定义张量流式多处理器”，丛书：国际计算机体系结构研讨会，2022。[15] N. P. Jouppi 等，“张量处理单元在数据中心内的性能分析”，载于第44届年度国际计算机体系结构研讨会论文集，2017。[16] S. Rivas-Gomez 等，“探索将视觉处理单元用作推理协处理器”，载于国际并行与分布式处理研讨会专题会议，2018。[17] Y.-C. Lin、B. Zhang 和 V. Prasanna，“Hp-gnn：在 CPU-FPGA异构平台上生成高吞吐量图神经网络训练实现”，载于国际现场可编程门阵列研讨会，2022。[18] M. Fey 和 J. E. Lenssen，“使用 PyTorch Geometric 快速进行图表示学习”，载于 ICLR图与流形表示学习研讨会，2019。[19] S. Gandhi 和 A. P. Iyer，“P3：大规模分布式深度图学习”，载于第15届USENIX操作系统设计与实现研讨会（OSDI 21），2021。[20] Z. Lin、C. Li、Y. Miao、Y. Liu 和 Y. Xu，“Pagraph：通过计算感知缓存扩展大规模图上的图神经网络训练”，载于第11届ACM云计算研讨会论文集，2020。[21] D. Zheng 等，“Distdgl：面向十亿规模图的分布式图神经网络训练”，载于2020年IEEE/ACM第10届不规则应用：架构与算法研讨会（IA3），2020。[22] W. Hu、M. Fey、M. Zitnik、Y. Dong、H. Ren、B. Liu、M. Catasta 和 J. Leskovec，“开放图基准：图机器学习数据集”，arXiv 预印本 arXiv:2005.00687，2020。[23] T. N. Kipf 和 M. Welling，“基于图卷积网络的半监督分类”，载于国际学习表征会议，2017。[24] J. Gilmer、S. S. Schoenholz、P. F. Riley、O. Vinyals 和 G. E. Dahl，“用于量子化学的神经消息传递”，CoRR，第abs/1704.01212卷，2017。[25] H. Zhang 等，“理解图神经网络计算图：从协同计算、I/O和内存的角度出发”，载于《机器学习与系统论文集》，2022。[26] J. Chen、R. Monga、S. Bengio 和 R. Jozefowicz，“重新审视分布式同步随机梯度下降”，载于国际学习表征研讨会，2016。[27] “同步随机梯度下降 [在线],” https://caffe2.ai/docs/SynchronousSGD.html，访问日期：2022-07-04。[28] 英特尔，“英特尔®快速通道互连简介 [白皮书],”，2009。[29] H. Zeng、H. Zhou、A. Srivastava、R. Kannan 和 V. Prasanna，“Graph-SAINT：基于图采样的归纳学习方法”，载于国际学习表征会议，2020。[30] M. Yan 等，“HyGCN：采用混合架构的图卷积网络加速器”，载于 IEEE国际高性能计算机体系结构研讨会，2020。[31] V. Kathail，“赛灵思 Vitis 统一软件平台”，载于2020年ACM/SIGDA国际现场可编程门阵列研讨会论文集，2020，第173–174页。[32] H. Zhou 等，“面向 FPGA 上高性能时序图神经网络推理的模型与架构协同设计”，载于国际并行与分布式处理研讨会，2022。[33] K. Huang、J. Zhai、Z. Zheng、Y. Yi 和 X. Shen，“理解并弥合当前图神经网络性能优化中的差距”，载于第26届ACM SIGPLAN并行编程原理与实践研讨会，丛书：PPoPP ’21，2021。[34] X. Chen 等，“Rubik：用于高效图神经网络训练的分层架构”，《IEEE集成电路与系统计算机辅助设计汇刊》，2021。[35] D. Zheng 等，“面向十亿规模异构图的图神经网络分布式 CPU 与 GPU 混合训练”，载于ACM SIGKDD知识发现与数据挖掘会议，2022。[36] Z. Cai 等，“DGCL：用于分布式图神经网络训练的高效通信库”，载于第16届欧洲计算机系统会议论文集，丛书：EuroSys ’21，2021。